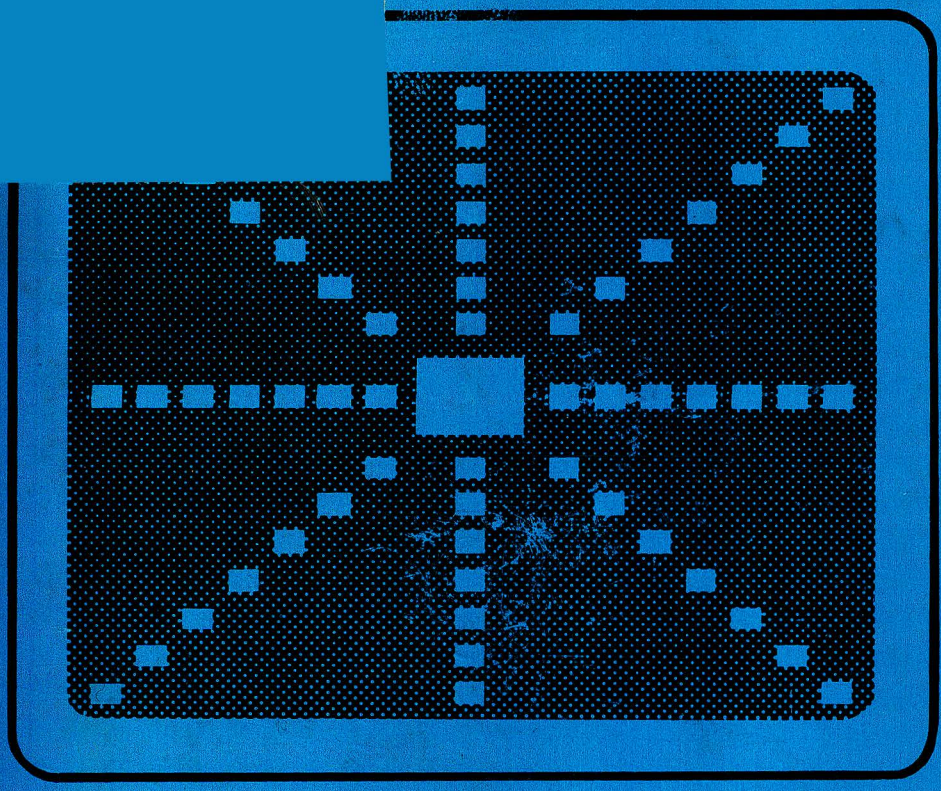
RPNL



Eine FORTH-ähnliche Sprache mit strukturunterstützenden Sprachkonstrukten

Gustav Wostrack

LV

R P N L

Eine
FORTH - ähnliche Sprache
mit
srukturunterstützenden Sprachkonstrukten

R P N L

Eine
FORTH - ähnliche Sprache
mit
srukturunterstützenden Sprachkonstrukten

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Wostrack, Gustav:
RPNL : e. FORTH-ähnl. Sprache mit struktur=
unterstützenden Sprachkonstrukten / Gustav
Wostrack. - Sprendlingen : Luther, 1984.
ISBN 3-88707-022-4

Alle Rechte, auch die der Übersetzung in fremde Sprachen, vorbehalten. Kein Teil dieses Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Die Nutzung der Programme ist nur für private Zwecke erlaubt. Die gewerbliche Nutzung der Programme bedarf der ausdrücklichen Genehmigung des Verlages.
Der Verlag übernimmt keine Gewähr dafür, daß beschriebene Programme, Schaltungen, Baugruppen, Verfahren etc. funktionsfähig und frei von Schutzrechten Dritter sind. Diverse, nicht näher bezeichnete Marken- und Firmennamen sind eingetragene Waren- oder Markenzeichen verschiedner Firmen.
©Copyright 1983 by W.-D. Luther - Verlag, 6555 Sprendlingen
Printed in Germany ** ISBN 3-88707-022-4 ** DM 49,80

INHALT

		Seite
Vorwort		7
1.	Was ist RPNL ?	9
2.	Der Compiler	12
2.1	Die Speicherorganisation	14
2.1.1	Die Stack	14
2.1.2	Die Namensliste	14
2.2	Das Laufzeitsystem	14
2.3	Die Kernprogramme	18
2.3.1	Konstanten	18
2.3.2	Variablen	18
2.3.3	Das Array	19
2.3.4	Der zusammengesetzte Typ	20
2.4	Die Compilerliste	20
2.5	Generatoren für die Programmtypen	23
2.5.1	CALL	24
2.5.2	CODE	25
2.6	Das Programmiersystem	26
2.6.1	Der Dialog	26
2.6.2	Die Systembefehle	27
3.	Die höhere Programmiersprache RPNL	30
3.1	Die Daten	30
3.1.1	Die Datentypen	30
3.1.2	Variablen und Konstanten	31
3.1.3	Standardnamen	32
3.2	Aufbau der Programme	32
3.2.1	Die Deklaration	32
3.2.2	Die Programmklammer mit Namen	34
3.2.3	Die Anweisung	34
3.2.3.1	Einfache Anweisungen	35
3.2.3.2	Ein erstes Programmbeispiel	41
3.2.3.3	Die Ein-und Ausgabeanweisungen	45
3.2.3.4	Anweisungen zur Steuerung der Anweisungen mit Beispielen	48
3.2.4	Die Diskettenbefehle	60
3.2.5	Anwendung der Diskettenbefehle	66
3.2.5.1	Erstes Beispiel (EINLESEN)	66
3.2.5.2	Zweites Beispiel (KOPIERE)	67

2.5.3 Drittes Beispiel (DATEN) 68
3.2.5.4 Viertes Beispiel (NAMEN) 69

3.3 Funktionen und Prozeduren in RPNL 71

3.4 Rekursionen in RPNL 73
3.4.1 Berechnung von Binomialkoeffizienten 77
3.4.2 Der Turm von Hanoi 80

4. Anwendungen der Programmiersprache RPNL 83

4.1 Sortieren eines Array 83
4.1.1 Das Programm 84
4.1.2 QUICKSORT 85

4.2 Die Erzeugung eines Magisches Quadrat 91

4.3 Test von Schreiblesespeichern 95

4.4 Implementierung eines Editors 97
4.4.1 Das Menue 97
4.4.2 Die Prozeduren 97
4.4.2.1 Die Eingabe 97
4.4.2.2 DRUCKE 98
4.4.2.3 LISTE 99
4.4.2.4 LOESCHE 99
4.4.2.5 SUCHE 100
4.4.3 Das Programm 100

5. Der RPNL - Compiler für Z-80 Mikrocomputer .. 106

5.1 Das Assemblerlisting 106
5.2 Das Hexdump 172

Anhang 1 Syntaxdiagramm der wichtigsten Befehle und Strukturen 180

Anhang 2 Liste der Schlüsselworte und Standardnamen... 184

Anhang 3 Literatur 185

Anhang 4 RPNL auf einen Blick 186

Vorwort

Diese Arbeit ist in mehrere Teile gegliedert. Der erste gibt zunächst eine Einführung in die Arbeitsweise des Compilers. Direkt damit verbunden ist ein Überblick über die Besonderheiten der Sprache, insbesondere der expliziten Benutzung des Stapelspeichers.

In die darauf folgende Erläuterung der Syntax und der Semantik wurden gleich einige praxisnahe Beispiele eingestreut. Dabei wurde darauf geachtet, daß Begriffe und Definitionen eindeutig gefaßt und benutzt werden.

Gerade zusammen mit dem vollständigen Assemblerlisting, das man in 3. Teil der Arbeit findet, erst werden alle Aspekte von RPNL in der Praxis erschöpfend ausgetestet werden können und dabei die überlegenen Merkmale zu Tage treten, die darin bestehen, durch maschinennahes Programmieren effiziente Programme auf Mikrocomputern zu realisieren.

Notizen:

1. Was ist RPNL ?

RPNL basiert auf der Sprache FORTH und hat gleichzeitig wesentliche Elemente aus PASCAL übernommen. Beispielsweise ist ihnen der Verzicht bzw. der sparsame Gebrauch von Sprunganweisungen gemeinsam. So besitzt RPNL alle die Sprachelemente, die eine strukturierte Programmierung unterstützen. Der wesentliche Unterschied zu anderen Programmiersprachen besteht in der

Verwendung der Umgekehrten Polnischen Notation

(UPN bzw. "R"everse "P"olish "N"otation), von der sich auch der Name der Sprache ableitet (das "L" steht für language). Mag diese ungewöhnliche Schreibweise im ersten Moment auch erschwerend wirken, so ermöglicht sie aber doch den Aufbau eines Compilers mit einem

Platzbedarf von lediglich 8 KByte.

Zu dieser Besonderheit kommt eine weitere hinzu. Alle Compiler erzeugen ein ausführbares Programm. Entweder in Maschinensprache, die direkt von der CPU verstanden und ausgeführt wird oder aber einen Interpreterkode, eine interne Darstellung des Programmes, die dann durch ein anderes Programm interpretiert/übersetzt und gleichzeitig ausgeführt wird. Im Gegensatz dazu

erzeugt RPNL einen Zwischenkode,

der kein Interpreterkode ist, der aber auch von der CPU nicht direkt ausgeführt wird. Vielmehr besteht der Kode aus einer Aneinanderreihung von Unterprogrammaufrufen. Dies sind Standardprozeduren, die eine Art Bibliothek darstellen und mit deren Hilfe neue Programme formuliert und erzeugt werden können. Die eigentliche Besonderheit liegt in dem

Verzicht auf den CALL-Aufruf.

Dadurch reduziert sich der Kode auf eine 2 Byte Adresse. Allerdings wird dadurch ein

Interner Übersetzer

notwendig, der die Adressen auswertet und als Unterprogramme ausführt. Da der Zwischenkode in sehr enger Beziehung zur Maschinensprache steht, sind

die Ausführungszeiten enorm kurz.

Insbesondere dann, wenn man die durch RPNL erzeugten Programme mit Interpreterprogrammen vergleicht, schneidet RPNL hervorragend ab.

In dieser Sprache hat jedes Programm, Unterprogramm und jedes Datum seinen Namen. Der Name für den Compiler selbst ist

das Schlüsselwort PROGRAM,

wodurch dann eine neue Definition bzw. ein Programm eingeführt wird. Diesem Schlüsselwort folgen zunächst der Name der Funktion oder Prozedur, die definiert werden soll, dann der Anweisungsteil und schließlich

das Schlüsselwort END,

das die Definition abschließt. Wie schon angedeutet, läßt RPNL keine bedingten oder unbedingten Sprünge zu. Dagegen verarbeitet der Compiler

das Bedingungsstatement IF-ELSE-THEN und
die Schleifen FOR-LOOP und
REPEAT-LOOP.

Alle drei Anweisungen sind aus der Sprache PASCAL bekannt und diesen auch sehr ähnlich. Weil alle Funktionen dieser Sprache ihre Argumentwerte von einem Stapelspeicher erwarten, sind auch diese Standardstatements in UPN geschrieben. RPNL verarbeitet 16-Bit-Integerzahlen einmal als Daten, aber auch als logische Werte. Dabei ist dem Wert "falsch" die Null und "wahr" eine Zahl ungleich der Null zugeordnet. Bevor allerdings einer Variablen oder Konstanten ein Wert zugewiesen werden kann, muß diese durch Nennung ihres Namens deklariert werden. Dies geschieht u.a. durch die

Schlüsselworte VAR bzw. CONST.

Insgesamt sind in RPNL 4 Datentypen zu unterscheiden. Diese sind:

* Integer,
* Boolean,
* Array und
* String.

Diese unterscheiden sich weniger in der Struktur und Leistung als eher in einer etwas abweichenden Formulierung, die wiederum durch die UPN bedingt ist, von den bekannten gleichnamigen Datentypen anderer Sprachen. Auf diese Datentypen sind Standardoperationen und Standardprozeduren definiert, die es mit Hilfe der übrigen Schlüsselworte gestatten, auch komplexe Programme in einer eleganten und übersichtlichen Weise zu formulieren. Zusammenfassend kann man die wesentlichen Vorteile der Sprache RPNL etwa folgendermaßen formulieren:

1. Das Spektrum der semantischen Möglichkeiten ist sehr weit: Von Hochsprachelementen, die eine strukturierte Programmierung ermöglichen, bis hin zu einfachen arithmetischen Funktionen.

2. Durch ein wirkungsvolles Unterprogrammkonzept, bei dem alle Funktionen und Prozeduren Namen tragen, können Programmodule erzeugt werden, die übersichtlich zum Gesamtprogramm zusammengesetzt werden können. Die Programmodule können aber getrennt erzeugt und getestet werden.

3. Der von dem RPNL-Compiler erzeugte Kode ist in der Ausführungszeit gewöhnlichen Interpretern überlegen.

4. Der Compiler ist kein unüberschaubares Mammutgebilde. Er ist dagegen handlich und kann den eigenen Bedürfnissen und Vorstellungen leicht angepaßt werden.

2. Der RPNL-Compiler

Nachdem nun die Programmiersprache RPNL im Überblick vorgestellt wurde, soll im folgenden Kapitel erläutert werden, wie ein dem Rechner übermitteltes Programm verarbeitet wird. Erst anschließend werden Syntax und Semantik der Sprache ausführlich dargestellt.

Die Aufgabe besteht also darin, einen lesbaren Programmtext in ein für die Maschine verständliches, also lauffähiges Programm zu überführen. Die verwendete Quellsprache ist dabei RPNL. Das Programm, das diese Umsetzung leistet, ist der COMPILER. Ein Compiler besteht im allgemeinen aus mehreren Teilen, die einander zuarbeiten. Im wesentlichen sind dies 3 Einheiten. Das Quellprogramm liegt oftmals als eine Folge von ASCII-Zeichen vor, die entweder im Speicher abgelegt sind oder aber während des Compiliervorganges über die Tastatur eingegeben werden. Das Programm, das diese Zeichen einliest, ist der SCANNER. Seine Aufgabe besteht also darin, eine lexikographische Analyse durchzuführen. Im einfachsten Falle besteht sie darin, den Fluß von Eingabesymbolen dann zu unterbrechen, wenn ein vollständiges Wort eingelesen wurde.

Anschließend reicht er diese Zeichenfolge an die nächste Einheit, den PARSER weiter. Dieser Algorithmus prüft die Folge von Eingabewörtern auf syntaktische Richtigkeit. Darunter ist eine Kontrolle zu verstehen, die nur implementierte Schlüsselworte und Standardnamen oder aber deklarierte Namen als korrekte Eingabe akzeptiert. Darüber hinaus stellt das Programm alle Fehler fest, die gegen die Ordnung verstoßen, die im Syntaxdiagramm festgelegt ist.

Als letzte Einheit ist der Vorgang zu nennen, der dafür sorgt, daß aus der Anweisungsfolge ein ausführbarer Kode erzeugt wird. Oftmals werden die beiden letztgenannten Einheiten, also die Syntaxanalyse und die Generierung des Objektkodes zu einer Einheit zusammengefaßt. Dies geschieht aus rein ökonomischen Gründen. Deshalb kommt dieses Verfahren auch bei dem vorliegenden RPNL-Compiler zur Anwendung. DarÜber hinaus ist zur Syntaxanalyse zu sagen, daß sich die Überprüfung auf die korrekte Benutzung der Standardnamen und Schlüsselworte beschränkt. Der Compiler ist also ein Programm, das die oben beschriebenen Aufgaben durchzuführen hat. Es wird in unserem Falle aufgerufen durch die Eingabe des Schlüsselwortes PROGRAM.

Wenn dann ein neuer Name eingegeben wird, der compiliert werden soll, so gelangt dieser zusammen mit anderen Daten (nämlich dem Programm) zunächst in einen Eingabepuffer, wo er Teil eines Datenstromes ist, den dieses Programm analysieren muß (Bild 1). Dies geschieht im wesentlichen durch Nachschlagen in einer Liste, in der die Namen von bereits definierten Prozeduren und Funktionen mit den zugehörigen Adressen stehen. Wird der Name dort gefunden, dann wird entweder diese Adresse gespeichert oder das zugehörige Programm ausgeführt. Das Compilieren eines neuen Na-

mens kann also ein relativ langer Vorgang sein, der etliche Schritte in Anspruch nimmt. Dazu noch etwas mehr. Für die meisten Anweisungen (einfachen Anweisungen) besteht die Aufgabe des Compilers einfach darin, die Adresse der zugehörigen Routine als Zwischenkode zu speichern. Tauchen Zahlen in einem Programm auf, dann werden diese so abgspeichert, daß sie während der Ausführung auf den Stack gelegt werden (durch eine PUSH-Instruktion). Für die restlichen Anweisungen ist dies nicht so einfach möglich. Sie verlangen während des Compilierens die Ausführung von speziellen Aktionen, wie Adressberechnungen oder Umspeicherungen.Das Compiler-Flag wird dann ungleich Null, wenn das Schlüsselwort END erkannt wird oder aber ein Fehler im obigen Sinne vorliegt.

```
PROGRAMM PROGRAM
     SETZE das Compilerflag auf Null
     LIES den Programmnamen
     SPEICHERE den Programmnamen
               und die Adresse in der Namensliste
     SPEICHERE einen Aufruf als erstes Wort
               im Kodebereich (Parameterfeld)
     WIEDERHOLE
          UNTERSUCHE das nächste Wort          ; der SCANNER
          WENN es in der Compilerliste steht, ; der PARSER
               DANN führe den zugehörigen Kode aus,
                                               ; der Kodegenerator
          SONST
               WENN es in der Interpreterliste steht,
                    DANN speichere seine Adresse im Kodebereich
               SONST
                    WENN es eine Zahl ist,
                         DANN speichere eine PUSH-Instruktion und
                              speichere die Zahl
                    SONST liegt ein Fehler vor,
                    ZEIGE Fehler an,
                    SETZE Zeiger auf Anfangswerte,
     BIS das Compiler-Flag ungleich Null ist,
ENDE.
```

Bild 1 **Der RPNL-Compiler als Programm in einer umgangssprachlichen Formulierung.**

Zusammenfassend kann man sagen, daß das Ergebnis eines Compiliervorganges immer darin besteht, daß ein neuer Name eingeführt und ein zugehöriges Programm generiert wurde.

In den nachfolgenden Abschnitten werden nun beschrieben:

* die Aktionen, die den Ablauf des Compiliervorganges steuern,
* die Struktur und die programmtechnische Realisierung der Programmtypen und
* die Verarbeitung einiger wichtiger Datentypen.

2.1 Die Speicherorganisation

2.1.1 Der Stack

Ein RPNL Programm wird so ähnlich wie ein Maschinensprachepro-gramm behandelt. Weil der eigentliche Stack so stark beansprucht wird, ist ein zweiter für die Rücksprungadressen vorgesehen. Ihr ständiger Gebrauch ist eines der Merkmale dieser Sprache. Der Stack ist der Ort, an dem Daten, Adressen und Flags abgespeichert werden, um sie dann wieder leicht zu erreichen, wenn sie von einem Teil des Programmes einem anderen übergeben werden sollen. Man könnte ihn aber auch als einen Akkumulator bezeichnen. Daten, die normalerweise in den Akku fließen, landen hier auf dem Stack. Softwareroutinen führen die üblichen Akkumulatorfunktionen durch. Die Ergebnisse - Daten, Flags oder auch Adressen - werden dort wieder abgelegt für den weiteren Gebrauch des Programmes.

An dieser Stelle soll einmal kurz der Begriff Stack erklärt werden. Als Stack bezeichnet man eine lineare Liste, bei der das Einfügen (mit der Operation PUSH) und Entfernen (mit POP) von Elementen nur von einem Ende der Liste möglich ist. Ein sogenannter Stackpointer zeigt jeweils auf den Speicherplatz, wo momentan gerade Manipulationen möglich sind. Da der Stack im allgemeinen nach unten, d.h. zu den niedrigeren Adressen wächst, muß der Stackpointer entsprechend bei POP erhöht und bei PUSH erniedrigt werden.

2.1.2 Die Namensliste

Jeder Routine des Compilers ist ein Label bzw ein Name zugeordnet. Mit diesem Namen kann dann das zugehörige Programm aufgerufen und ausgeführt werden. Neue Aufgaben können durch eine sinnvolle Kombination von bereits eingeführten Namen gelöst werden. Diese Sequenz wird dann durch ihren eigenen Namen identifiziert. Um RPNL diesbezüglich in seiner Buchhaltung zu unterstützen, wird jeder dieser Namen in einer Namensliste abgespeichert (Bild 2). Sie wird benötigt, um die Startadresse einer aufgerufenen Routine wiederzufinden. Da die Namen normalerweise von unterschiedlicher Länge sind, wird zu deren Abspeicherung ein festgelegtes Format benutzt. Dabei gibt das erste Byte die Länge des Namens an, also die Anzahl von Zeichen, die den Namen gerade ausmachen. Wird ein Wort gesucht, so vergleicht RPNL zunächst nur die Länge der beiden Kandidaten. Erst wenn diese übereinstimmt, werden die Namen zeichenweise verglichen. So ergibt sich ein relativ schneller Suchalgorithmus.

2.2 Das Laufzeitsystem

RPNL-Programme haben einen charakteristischen Aufbau: Sie bestehen aus einem Kodefeld und einem Parameterfeld (Bild 3). Vom Programmiersystem aus und während des Compiliervorganges wird mithilfe der Namensliste auf diese Kernprogramme, die auch Primi-

```
                                                                  hohe Adresse
-------------
Compiler-           I 0 I        I         I
liste               I 6 I REPEAT I Adresse I>-------------------
                    I 5 I WRITE  I Adresse I>----------------- l
------------        I 3 I VAR    I Adresse I>--------------- l l
Inter-              I 0 I        I         I               l l l
preter-             I 3 I DIV    I Adresse I>------------- l l l
liste               I 4 I SWAP   I Adresse I>----------- l l l l
                    I 2 I OR     I Adresse I>--------- l l l l l
------------        ------------------------         l l l l l l
eigene              I 4 I TEST   I Adresse I>---     l l l l l l
Namen               I 5 I WERT1  I Adresse I>--l--   l l l l l l
                    I 5 I WERT2  I Adresse I>--l-l-- l l l l l l
------------        ------------------------   l l l l l l l l l
                    I    l                 I   l l l l l l l l l
freier              I    l                 I   l l l l l l l l l
Raum                I    l                 I   l l l l l l l l l
für                 I    V                 I   l l l l l l l l l
Programmme          I                 l    I   l l l l l l l l l
                    I                 l    I   l l l l l l l l l
                    I                 l    I   l l l l l l l l l
                    I Kode für TEST   l    I<--- l l l l l l l l
------------        ------------------------     l l l l l l l l
                    I                      I     l l l l l l l l
                    I                      I     l l l l l l l l
für                 I                 l    I     l l l l l l l l
Daten               I Kode für WERT2  l    I<----l-- l l l l l l
                    I Kode für WERT1  l    I<-----   l l l l l l
------------        ------------------------         l l l l l l
                    I Kode für OR          I<--------- l l l l l
Routinen            I Kode für SWAP        I<----------- l l l l
                    I Kode für DIV         I<------------- l l l
des                 I Kode für VAR         I<--------------- l l
                    I Kode fÜR WRITE       I<----------------- l
Compilers           I Kode für REPEAT      I<-------------------
                    I                      I
                    I weitere Routinen     I
                    I                      I
                                                  niedrige Adresse
```

Bild 2 **Prinzipielles Zusammenwirken der Listen im RPNL-Compiler**

tive genannt werden, zugegriffen. Im Gegensatz dazu wird während der Laufzeit eine andere Art des Zugriffs benutzt. Dann wird nicht mehr der Name, sondern lediglich die zugehörige Adresse benötigt. Der Zugriff ist dann insbesondere Aufgabe eines Interpretations-Pointers (IP) und einer NEXT-Routine. Der IP ist etwa vergleichbar mit dem Programmzähler einer CPU. Und NEXT ist die zugehörige Software-Routine (Bild 4). Jeder Programmteil endet mit einem Sprung zur NEXT-Routine. Sie inkrementiert den IP und leitet damit sofort die Ausführung des nächsten Programmschrittes

```
                                                  -------------- -
                                  --------->  I  Kodefeld  I  l
                                 l            --------------  l
                                 l            I            I  l
                                 l            I Parameter- I   ->Kernpro-
                                 l            I   feld     I  l  gramm
                                 l            I            I  l
                                 l            -------------- -
                                 l    ----->  I  Kodefeld  I  l
                  I        I  l   l           --------------  l
------            ----------  l   l           I            I  l
I IP I -------->  I Name A I -    l           I Parameter- I   -> Kernpro-
------            ----------      l           I   feld     I  l  gramm
                  I Name B I --   l           I            I  l
                  ----------   l  l           -------------- -
                  I Name C I --l--    -->     I  Kodefeld  I  l
                  ----------   l     l        --------------  l
                  I Name D I --l-----         I            I  l
                  ----------   l              I Parameter- I   -> Kernpro-
                  I        I   l              I   feld     I  l  gramm
                      I        V              I            I  l
                  Namenszeiger                -------------- -
```

Bild 3 **Die Namenszeiger verweisen auf die zugehörigen Kernprogramme**

```
                  I        I
------            ----------      --------------
I IP I -------->  I Name G I --->  I  Kodefeld  I -------------
------            ----------      --------------    -------   l
  l               I Name H I --    I Maschinen- I <--I     I---
  l               ----------   l   I  sprache-  I    -------
  l               I        I   l   I programm   I      PC
  l                            l   I            I
  l                        (3) l   ............
  l                        ----l->I  Jump NEXT I
  l                      l     l   --------------
   ----------------      l     -> I  Kodefeld  I -------------
   (2) IP:=IP+1     l    l        --------------    -------   l
                    l    l(1)     I            I <--I     I---
                    l    l        I            I    -------
                    l    l                            PC'
                    l    V
---------           -------------------------
I       I ----->    I       N  E  X  T       I
---------           -------------------------
   PC'
```

Bild 4 **Die NEXT-Routine im zeitlichen Ablauf**

ein. Danach zeigt der IP auf die nächste aktuelle Speicherzelle. Diese enthält dann die Adresse des Kodefeldes des nächsten Befehles. Kodefeld deshalb, weil von dort immer auf einen Speicher-

platz verwiesen wird, wo Maschinenkode abgelegt ist, den es auszuführen gilt (Bild 5).

Wenn IP wieder inkrementiert wird, ist damit die nächste auszuführende Adresse festgelegt. So werden durch Erhöhen von IP genau die Programmteile ausgeführt, die der Programmablauf verlangt. Die Speicherzelle, auf die der IP zeigt, wird auch Namenszeiger genannt, weil durch ihn das zu dem Namen gehörige Programm ausgewählt wird, und er somit als Zeiger fungiert. Man vergegenwärtige sich, daß ein Zeiger die Adresse dessen beinhaltet, auf was er zeigt. Der Inhalt des Parameterfeldes definiert den Namen bzw die zugehörige Aktion. Der Inhalt des Kodefeldes wird diktiert von dem Inhalt des Parameterfeldes. Das soll sagen, daß es zu jedem Typ von Parameterfeld eine korrespondierende Adresse im Kodefeld existiert, und deren zugehörige Routine angibt, wie das Parameterfeld zu interpretieren ist. So kann ein Parameterfeld folgenden Inhalt haben:

* ein Programm in Maschinensprache
* eine Variable
* eine Konstante
* eine Liste von bereits definierten Namen.

```
                I        I
------          ----------      --------------
I IP I -------> I Name M I ---> I  Kodefeld  I -------------
------          ----------      --------------    -------   1
                I        I      I Parameter- I <--I     I---
                            --> I    feld    I    -------
            Maschinen- )  1     I            I      PC
             sprache-  )--      I            I
            programm   )         ............
                                I  Jump NEXT I
                                --------------
```

Bild 5 **Aufbau der Kernprogramme**

Dementsprechend lautet dann auch der Inhalt des Kodefeldes. Es wurde gezeigt, daß die NEXT-Routine die Ausführung der Kernprogramme während der Laufzeit steuert. Man spricht deshalb auch vom Laufzeitsystem. Da diese Routine zusätzlich zu jedem Teilprogramm ausgeführt wird, ist die Forderung naheliegend, das Laufzeitsystem bezüglich der Ausführungszeit zu optimieren. Dies ist im vorliegenden Falle auch geschehen. Andererseits bietet sich aber hier die Möglichkeit, Kontrollmechanismen einzubauen, die den ordnungsgemäßen Programmablauf überwachen. Beispielsweise kann ständig die benutzte Stacktiefe übeprüft werden. Dies ist insbesondere bei rekursiven Programmen empfehlenswert. Man muß sich dabei aber vor Augen halten, daß dies immer auf Kosten der Ausführungszeit geht.

2.3 Die Kernprogramme

Wenn das Parameterfeld des Namens ein Maschinenspracheprogramm enthält, steht im Kodefeld des Namens seine eigene Parameterfeldadresse (PFA). Wenn NEXT den Programmzähler mit dem Inhalt der Parameterfeldadresse lädt, wird das Maschinenspracheprogramm ausgeführt. Der letzte Befehl des Programmes lädt den Programmzähler mit der Startadresse von NEXT. Als Beispiele für solche Programme wird nachfolgend die Verarbeitung verschiedener Prozeduren gezeigt.

2.3.1 Konstante

Bei der Repräsentation einer Konstanten besteht das Parameterfeld lediglich aus zwei Byte, die deren Wert beinhalten (Bild 6). Das

```
        I        I      I        I       PC       ------------
        ----------      ----------     ------     I CONSTANT I
    --> I ALPHA  I ---> ICONSTANTI --> I    I --> I          I
   l    ----------      ----------     ------     I Routine  I
   l    I        I      I (ALPHA)I --             I          I
   l                    ----------   l            I          I
-------                 I        I   l            ------------
I     I                              l
-------                              l               Stack
  IP                                 l            ------------
                                      ----------> I          I
                                                  ------------
                                                  I          I
                                                  ------------
                                                  I          I
```

Bild 6 **Das Zusammenwirken der Zeiger bei der Verarbeitung von Konstanten**

Kodefeld enthält eine Adresse, die immer (bei allen Konstanten) gleich ist und auf ein Maschinenspracheprogramm zeigt, das CONSTANT heißt. Wenn von NEXT diese Adresse in den Programmzähler geladen wird, wird der Inhalt des Parameterfeldes auf den Stack gelegt. Anschließend verzweigt das Programm wieder zu NEXT.

2.3.2 Variablen

Ähnlich ist es im Falle einer Variablen (Bild 7). Dort besteht das Parameterfeld lediglich aus aus einem 2 Byte Wort, das dessen Wert repräsentiert. Allerdings variiert die Funktion des zugehörigen Programmes, das hier VARIABLE heißt. Denn wenn NEXT dieses Programm ausführt, wird nicht der Inhalt des Parameterfeldes, sondern ein Zeiger darauf, also seine Adresse (Parameterfeldadresse), auf den Stack gelegt. Mit der letzten Instruktion verzweigt die Routine wieder zu NEXT.

2.3.3 Arrays

Wurde ein Name als ARRAY deklariert, so ist das dazugehörige Parameterfeld genausoviele Worte lang, wie bei der Deklaration angegeben wurde (Bild 8). Die in dem zugehörigen Kodefeld stehende Adresse ist die gleiche wie die bei Variablen. Also legt das korrespondierende Programm die Adresse des ersten Wortes des Parameterfeldes auf den Stack. Durch Addition eines Offsetwertes kann auf jedes Wort innerhalb des ARRAY zugegriffen werden. Der eigentliche Zugriff erfolgt mit der Funktion '?' , die bei Eingabe einer Adresse, den dazugehörigen Wert abliefert bzw auf den Stack legt.

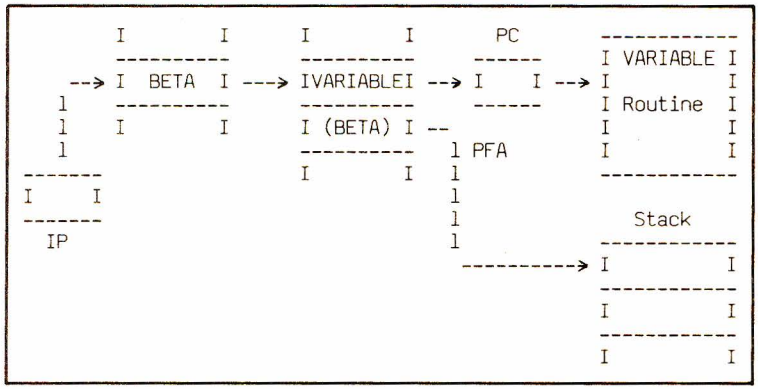


Bild 7 Das Zusammenwirken der Zeiger bei der Verarbeitung von Variablen

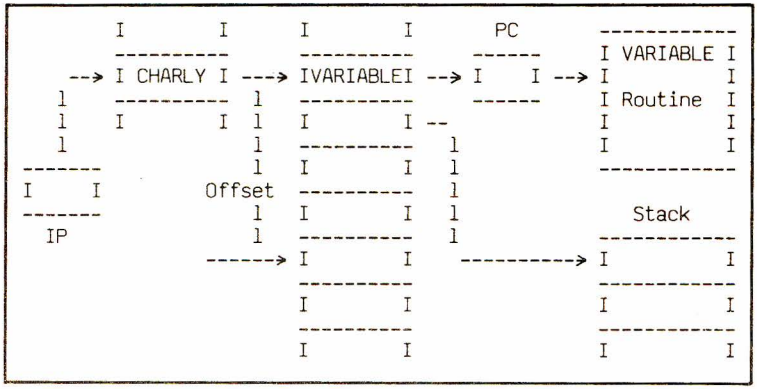


Bild 8 Der Aufbau von Arrays mit Hilfe von Variablen

2.3.4 Der zusammengesetzte Typ

Ein wesentliches Merkmal von RPNL ist, daß auf Namen, die bereits in die Namensliste eingetragen sind, immer wieder zugegriffen werden kann. Ein später definierter Name kann einen früheren zu seiner eigenen Definition benutzen. Also kann ein neuer Programmteil, d.h. ein Name alle alten so in geeigneter Weise wieder benutzen, daß die gestellte Aufgabe erfüllt wird. Für diese Art von Namen enthält das Kodefeld die Adresse eines Maschinenspracheprogrammes, das im wesentlichen zwei Aufgaben erfüllt (Bild 9). Einmal legt es den aktuellen Inhalt des inkrementierten IP

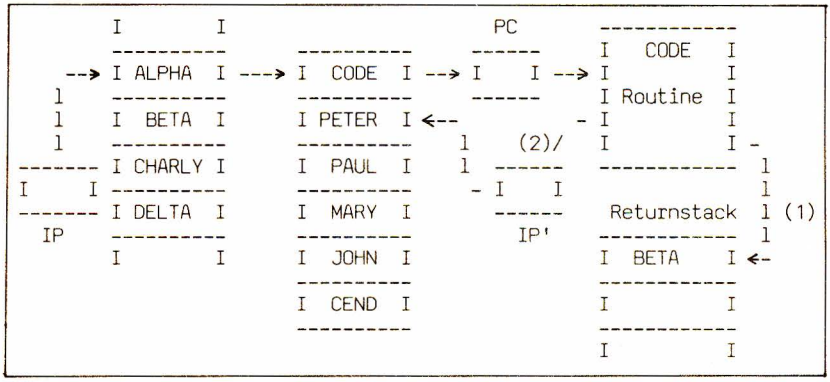


Bild 9 **Der zusammengesetzte Typ wird eingeleitet durch die Prozedur CODE**

auf den Returnstack. Gleichzeitig speichert es den Inhalt des Namenszeigers auf dem IP ab. Anschließend lädt das Programm die Adresse von NEXT in den Programmzähler. Im gezeigten Beispiel wird IP im Laufe der Abarbeitung von den Routinen, die zu ALFA gehören, ständig erhöht. Dieser Ablauf wird Verschachtelung (nesting) genannt. Programme können theoretisch beliebig häufig geschachtelt werden. Dieser Vorgang läuft ähnlich wie bei Maschinenspracheprogrammen ab. Der gesamte Programmteil wird abgeschlossen durch eine Routine, die den Rücksprung bewirkt, also die Funktion des oben erklärten Programmteils wieder aufhebt (Bild 10). Es lädt die oberste Adresse auf dem Returnstack in den IP und springt dann wieder zu NEXT. CODE und CEND sind zusammengehörige Hilfsprogramme, die diesen Typ von Namen, den wahrscheinlich häufigsten, überhaupt ermöglichen.

2.4 Die Compilerliste

Wenn also jeder verschiedene Namenstyp andere Adressen im Kodefeld verlangt, woher weiß dann der Compiler, welche Adresse er zu benutzen hat, wenn ein neuer Name definiert wird ? Dies geschieht durch die Deklaration. Der Programmierer entscheidet, welchen Typ er kreiert. Daraufhin agiert der Compiler entsprechend.

```
        ----------
        I  CODE  I
        ----------
        I PETER  I                                        I        I
        ----------                                        ----------
        I  PAUL  I                              IP        I ALPHA  I
        ----------                            ------      ----------
        I  MARY  I                            I    I --> I  BETA  I
        ----------                            ------      ----------
        I  JOHN  I        PC        ------------  l       I CHARLY I
        ----------      ------      I   CEND   I  l       ----------
    --> I  CEND  I --> I    I --> I           I  l       I DELTA  I
   l    ----------      ------      I Routine  I  l       ----------
   l                                I          I  l       I        I
   l                                I          I  l
-------                             ------------  l
I     I                                           l
-------                             Returnstack   l
  IP'                               ------------  l
                                    I  BETA    I -
                                    ------------
                                    I          I
                                    ------------
                                    I          I
```

Bild 10 **Die Prozedur CEND beendet ein Programm vom zusammengestzten Typ**

Schon ganz zu Anfang des Kapitels wurde erwähnt, daß einige Namen während des Compilierens andere Aktionen auslösen als während der Ausführung. Das ist auch sofort einleuchtend. Als Beispiel betrachten wir eine Variablendeklaration (Bild 11). Die Zeichenfolge VAR EGON bewirkt während des Compilierens folgende Aktionen:

* Es wird ein Kernprogramm angelegt. Das Kodefeld enthält die Adresse der VARIABLE-Routine; das Parameterfeld den notwendigen Speicherplatz (2 Byte).

* Der Name EGON wird in die Namensliste eingetragen. Die zugehörige Adresse zeigt auf das o.a. Kernprogramm.

Dagegen erzeugt die Nennung des Namens EGON während des Compiliervorganges lediglich den Eintrag der zugehörigen Adresse in den Kodebereich. Diese Adresse bewirkt dann während der Ausführungszeit, daß ein Zeiger auf das Parameterfeld der Variablen EGON auf den Stack gelegt wird. Dies erscheint einleuchtend. Aber woher weiß der Compiler, ob er während des Compilierens lediglich eine Adresse abzulegen hat oder aber mehrere, für die jeweilige Aktion ganz spezifische Befehle durchführen muß? Die Antwort darauf ist ganz einfach. Denn wie auch aus Bild 2 hervorgeht, benutzt der Compiler 2 Listen, nämlich

```
          I        I
          ----------                                   Variablenbereich
          I        I                             (1) I        I
          ----------      ------    ----------- ADR ----------
     -->  I  VAR   I --> I    I -->I   VAR   I --> IVARIABLEI
     l    ----------      ------    I         I     ----------
     l    I  EGON  I        PC      I Routine I -   I   00   I
     l    ----------                I         I  l  ----------
  ------- I        I                ----------  l  I        I
  I     I ----------                            l  ----------
  ------- I        I                            l  I        I
   IP                                      (2) l
                                               l  Namensliste
                                               l  I        I
     - während der                             l  ----------
       Compilezeit                             l  I        I
                                               l  ----------
                                                ->I  EGON  I
                                                  ----------
                                                  I   ADR  I
                                                  ----------
                                                  I        I
                                                  ----------
                                                  I        I


         I        I      I        I       PC       ------------
         ----------      ----------     ------     I VARIABLE I
     -->  I  EGON  I ---> IVARIABLEI --> I    I --> I          I
    l     ----------      ----------     ------     I Routine  I
    l     I        I      I   34   I                I          I
    l                     ----------                I          I
 -------                  I        I                ------------
 I     I
 -------                  - während der Ausführung
   IP
```

Bild 11 **Die Verarbeitung einer Variablendeklaration**

* die Compilerliste und
* die Namensliste.

In der Compilerliste stehen nun alle die Namen der Routinen, die schon während der Erzeugung des Objektkodes die Ausführung einer Programmfolge verlangen. Diese Liste kann vom Benutzer durch den Compiler nicht verändert werden. Die Einträge in der Namensliste dagegen existieren lediglich als Adressen. Sie werden in den Kodebereich abgelegt und bestimmen während der Ausführungszeit die Reihenfolge der Abarbeitung der gewünschten Routinen.

2.5 Generatoren für die Programmtypen

Mit dem RPNL-Compiler lassen sich zunächst nur Programme vom zusammengesetzten Typ erzeugen. Das sind also solche, die als Objektkode wiederum den Zwischenkode benutzen. Programmteile in Maschinensprache lassen sich bislang nicht einbauen. Bevor aber Programme vorgestellt werden, die genau dies leisten, sollen die Programmtypen nochmals definiert werden.

Typ 1: Maschinenspracheprogramm

Es wird von einem anderen Maschinenspracheprogramm mit einem CALL-Befehl, beispielsweise CD XX YY aufgerufen. Der Typ 1 selbst wird mit einem RETURN (z. B. C9) abgeschlossen.

Typ 2: Kernprogramm

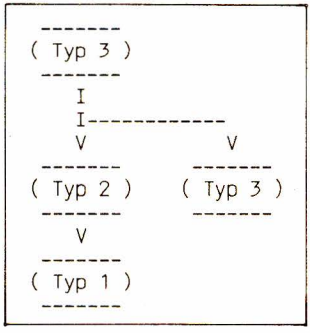
Es beginnt mit dem sogenannten Kodewort. Diesem folgt ein Programm in Maschinensprache. Das Kodewort selbst ist eine Adresse. Und zwar genau die der ersten Instruktion in Maschinensprache. Im Normalfall ist dies auch die Adresse des Kernprogrammes, um 2 erhöht. Der Typ 2 wird abgeschlossen durch einen Sprung zum Internen Übersetzer.

Typ 3: Der zusammengesetzte Typ

Dieser Typ enthält keine Anteile mehr an Maschinensprache. Er besteht ausschließlich aus Adressen, die zu anderen Programmen verweisen (Vektorliste). Diese Programme können dann vom Typ 2, aber auch vom Typ 3 sein. Die erste und die letzte Adresse der Programme dieses Typs sind stets spezielle CALL-und RETURN-Anweisungen. Dies ist deshalb notwendig, damit der Interne Übersetzer die Typen 2 und 3 unterscheiden kann. Denn beide Typen beginnen mit einer Adresse.

Die 3 Programmtypen sind in einer festgelegten Hierarchie geordnet. Das soll bedeuten, daß nicht jeder Typ jeden anderen aufrufen kann. Es sind aufgrund des Aufbaus folgende Aufrufe möglich:

V bedeutet: kann aufrufen:



Die Begründung hierfür wird dann klar, wenn man sich die Funktionsweise des Internen Übersetzers vor Augen hält: Zunächst erhöht er den Programmzähler, so daß dieser zur aktuellen Speicherstelle zeigt. Er greift deren erstes Wort (das ist immer eine Adresse) auf, springt dorthin und führt die entsprechende Instruktion aus. Dies ist beim Typ 2 ein Befehl in Maschinensprache und beim Typ 3 wiederum eine Adresse. Wie aber schon angedeutet, müssen alle Programme vom Typ 3 letzten Endes doch in ein Programm vom Typ 2 münden, damit der Prozessor die entsprechenden Instruktionen auch ausführen kann. Man vergegenwärtige sich, daß der Typ 1 mit dem Internen Übersetzer nie in Berührung kommt. Das gesamte Compilerprogramm setzt sich wie folgt aus den einzelnen Programmtypen zusammen:

ca. 80 %	Programmtyp 3
ca. 10 %	Programmtyp 2
ca. 10 %	Programmtyp 1

Aus der Beschreibung der Programmtypen erkennt man, daß der Typ 3 maschinenunabhängig ist, d.h. diese Programmteile sehen bei einem Z-80 Computer genauso aus wie bei einem 6502. Also: Wird der RPNL-Compiler (ursprünglich für Z-80 geschrieben) umgeschrieben für einen anderen Prozessortyp, dann sind lediglich etwa 20 % des Gesamtprogrammes neu zu erstellen. Der Rest sind Adressänderungen (die natürlich ein Assemblerprogramm erledigt). Letzten Endes bedeutet das sogar, daß ein Programm, das auf einem 6502 laufen soll, auf einem Z-80 erstellt werden kann.

2.5.1 CALL (Syntax: CALL NAME)

Das im folgenden beschriebene Programm ermöglicht es, den Programmtyp 1 einzugeben. Dabei besteht der Programmtext aus Sedezimalzahlen. Eine Eingabe in Assembler (Mnemonics) ist einer Erweiterung vorbehalten. Dazu sind im wesentlichen 3 Aktionen durchzuführen:

(1) Setzen von Anfangswerten
(2) Verarbeitung der Eingabe von HEX-Zahlen
(3) Aktualisierung der Systemvariablen

Die Aktion (1) wird von dem Programm INIT vollzogen.

```
PROGRAMM   INIT
           SETZE das Compiler-Flag auf Null
           LIES den Namen
           TRAGE ihn ein in die Interpreterliste
ENDE
```

Die Durchführung der Aktionen (2) und (3) ist Aufgabe des Programmes EINGABE.

```
PROGRAMM  EINGABE
    WIEDERHOLE
       LIES das nächste  Wort
       WENN  es eine HEX-Zahl ist,
          DANN trage es ein in  den Kodebereich
       SONST
           SUCHE das Wort in der Interpreterliste.
           WENN es gefunden wird,
              DANN trage seine Adresse ein,
           SONST melde Fehler.
           ENDE
       ENDE
    BIS das Compiler-Flag ungleich Null ist.
ENDE.
```

Insgesamt sieht CALL so aus:

```
PROGRAMM CALL
        INIT
        EINGABE
ENDE
```

Die Eingabe der Programmtypen 1 wie auch 2 wird abgeschlossen durch das Schlüsselwort CEND.

```
PROGRAMM CEND
        SETZE das Compiler-Flag auf ungleich Null
ENDE
```

2.5.2 CODE (Syntax: CODE NAME)

Das Programm CODE, das die Eingabe des Programmtypes 2 ermöglicht, ist im Aufbau dem Programm CALL ähnlich. Es geht lediglich insofern darüber hinaus, daß noch ein Sprung zum Internen Interpreter (NEXT) an den Schluß des Programmtextes gesetzt wird.

```
PROGRAMM CODE
     INIT
     TRAGE den 'Namen + 2' in den Kodebereich ein
     EINGABE
     TRAGE 'C3 NEXT' in den Kodebereich ein
     KORRIGIERE die Systemvariablen
ENDE.
```

Ein kurzes Anwendungsbeispiele für die Programmtypen 1 und 2:

```
CALL $ADD
                ; ADDITION DER KONSTANTEN 0010H AUF DEN INHALT
                ; DES REGISTERS HL
        16 00           ;       LD      D,00
        1E 10           ;       LD      E,10H
        19              ;       ADD     HL,DE
        C9              ;       RET
CEND
```

```
CODE ADD
                ; AUF DEN DIE BEIDEN OBERSTEN WERT DES STACK
                ; WIRD DIE KONSTANTE 10H ADDIERT
        E1              ;       POP     HL
        CD $ADD         ;       CALL    $ADD
        E5              ;       PUSH    HL
        C1              ;       POP     BC
        E1              ;       POP     HL
        CD $ADD         ;       CALL    $ADD
        E5              ;       PUSH    HL
        C5              ;       PUSH    BC
CEND

PROGRAM HAUPT
                :
                :
                :
        WERT1 ?
                WERT2 ?
                        ADD
                WERT2 :=
        WERT1 :=
                :
                :
                :
END
```

2.6 Das Programmiersystem

Der Compiler ist eingebettet in ein Rahmenprogramm, das neben dem Starten des Compilers auch die Ausführung von Hilfsroutinen und schließlich der Benutzerprogramme ermöglicht. Im nachfolgenden Abschnitt werden die dazu notwendigen Algorithmen und die zur Verfügung stehenden Prozeduren vorgestellt.

2.6.1 Der Dialog

Computer sollen so einfach wie möglich zu bedienen sein. Deshalb muß es ein Programm geben, daß den Dialog ermöglicht zwischen Anwender und Computer, d.h. die Wünsche des Anwenders dem Computer übersetzt. Dazu stehen mehrere Prozeduren in einem Betriebssystem zur Verfügung. Dieses Betriebssystem meldet sich zunächst mit "Ok!" und Übergibt die weitere Steuerung dann dem Programm INTERACT, dem eigentlichen Anwender-Übersetzer. Damit lassen sich weitere Systembefehle, aber auch Programme eingeben, compilieren und ausführen.

INTERACT

Dieses Programm wird beim Start des Compilers automatisch aufgerufen. Der Anwender gibt auf dem Eingabemedium Anwei-

sungen ein. Daraufhin untersucht das System die aktuelle Zeile nach folgendem Algorithmus:

```
PROGRAMM INTERACT
     LIES die Zeile
     WIEDERHOLE folgendes
          UNTERSUCHE das nächste Wort
          WENN es in der Namensliste steht,
               DANN führe den dazugehörigen Kode aus
          SONST
               WENN es eine Zahl ist,
                    DANN lege sie auf den Stack,
               SONST
                    GIB "Fehlermeldung" aus,
     BIS "Ende der Zeile" erkannt;
     SCHREIBE "Ok!";
ENDE.
```

(Der obige Algorithmus ist natürlich kein RPNL, sondern eine Vorstufe. Man sieht leicht, wie umgangssprachlich ein Programm formuliert werden kann.)

Es soll noch einmal erwähnt werden, daß das Betriebssystem keine Routine aus der Compilerliste aufrufen kann. Diese Liste ist dem Compiler vorbehalten (und selbst der kann sie lediglich lesen). Deshalb durchsucht INTERACT ausschließlich die Namensliste nach ausführbaren Programmen.

2.6.2 Die Systembefehle

Im folgenden Abschnitt werden die Befehle vorgestellt und erklärt, die es ermöglichen,

- RPNL-Programme dem Computer im Dialog mitzuteilen und
- daraus lauffähige Software zu compilieren.

NEW

Der Befehl NEW löscht alle eingegebenen Definitionen (Programme, Deklarationen) und setzt alle internen Zeiger und Zähler auf Anfangswerte. Im einzelnen sind dies:

- Beginn der Namensliste (Liste der Namen, die der Benutzer definiert),
- Beginn des compilierten Programms (das ist der Bereich, wo der Zwischenkode abgespeichert wird),
- Beginn des Returnstack (Stapelspeicher für die RÜcksprungadressen),
- Beginn des Parameterstack,
- Beginn des Variablen- und Konstantenbereiches.

LIST

Dieser Befehl unterstützt die Fehlersuche und die Programmerstellung allgemein. Mit ihm kann man einen TRACE-Programmlauf durchführen. Nach Eingabe des Befehls LIST werden alle bisher eingegebenen einfachen Zeiger mit ihren zugehörigen Adressen und aktuellen Werten im Dezimalsystem ausgegeben. Folgendes Beispiel zeigt die Wirkungsweise.

```
-Ok!
-NEW
-DECLARE
        VAR WERT1       VAR WERT2
 DEND
-Ok!
-PROGRAM TEST
        123  WERT1  :=
        321  WERT2  :=
 END
-Ok!
-LIST
WERT2                          6167          0
WERT1                          6163          0
TEST                           6912       4504
-Ok!
-TEST
-LIST
WERT2                          6167        321
WERT1                          6163        123
TEST                           6912       4504
-Ok!
```

Wir können aus diesem Beispiel mehrere Erkenntnisse schöpfen. Die erste Eingabe des Befehls LIST zeigt, daß die einfachen Zeiger TEST, WERT1 und WERT2 in die Namensliste eingetragen sind, und den Variablen WERT1 und WERT2 ist ihr Anfangswert (wie bei allen Variablen), nämlich die Null zugewiesen. Erst nach dem Programmlauf TEST zeigt der Ausdruck von LIST, daß die Speicherinhalte von WERT1 jetzt in 123 bzw von WERT2 in 321 geändert worden sind. Bei TEST ändert sich natürlich nichts. Will man einen TRACE-Lauf starten, so wird der Befehl LIST an geeigneter Stelle im Programm eingebaut. In unserem Beispiel könnte dies so aussehen:

```
PROGRAM TEST
        123  WERT1  :=
        321  WERT2  :=
        LIST
END
```

MEMORY

Nach Eingabe dieses Befehls wird der noch zur Verfügung stehende Speicherraum in Byte angezeigt. Auch diese Zahl wird dezimal angegeben.
Unter Speicherraum ist der derjenige Raum zu verstehen, der für compilierte Programme zur Verfügung steht.

SYSTEM

Dieser Befehl setzt den DMA-Bereich wieder zurück auf 80H, verläßt den RPNL-Compiler und springt zurück ins Betriebssystem, veranlaßt somit einen Warmstart.

SAVE NAME

Mit diesem Befehl sichern Sie ein Programm, das Sie gerade mit dem Compiler erfolgreich übersetzt haben. Dieses Programm ist

1. auf der Diskette im Betriebslaufwerk gespeichert und
2. unter NAME im CP/M Betriebsystem direkt aufrufbar. Der Zusatz ".COM" wird vom System angebracht.

Wenn dieses Programm keine Diskettenbefehle benutzt, sollten Sie vor dem Compilieren den Befehl NOFILES geben. Dadurch wird Ihr Programm auf der Diskette kürzer und ist auch schneller geladen.

FILES

Dieser Befehl entspricht in etwa dem DIR des CP/M und gibt das Directory der aktuellen Diskette aus.

Beispiel:

FILES

```
Dateien auf dem Laufwerk A:
  RPNL    .COM     E80     .COM     §MSK    .RPN     COPY    .COM
  VERS    .BRF     SERIE   .BRF     TEST    .TXT     BIOS    .MAC
```

DRON

Alle Ausgaben werden auf dem Drucker protokolliert.

DROFF

Schaltet DRON und somit den Drucker wieder aus.

Alle Befehle des Betriebssystems können Sie natürlich auch in Ihren Programmen verwenden. Eine Ausnahme bildet die Aktion SAVE.

Die von dem CP/M oder kompatiblen Systemen bekannten Steuerzeichen

CTRL P	Ein-/Auschalten des Druckers
CTRL S	Anhalten der Ausgabe
CTRL Q	Fortsetzen der Ausgabe

haben keine Gültigkeit mehr, da bei der Ausgabe direkt die Einsprünge in den BIOS benutzt werden, die wesentlich schneller sind.

Kommentare

Darüber hinaus besteht die Möglichkeit, mit dem Zeichen ';' (Semikolon) Kommentare in ein Programm einzubauen, d.h. alle Zeichen zwischen dem ';' und dem Zeilenende werden als Kommentar aufgefaßt und nicht übersetzt. Zu beachten ist dabei allerdings, daß zwischen dem ';' und dem ersten Zeichen des Kommentartextes mindestens ein Leerzeichen stehen muß.

Fehlermeldungen

Das System kann während der Verarbeitung folgende Fehlermeldungen ausgeben:

- Not defined: WORT
  Sie haben versucht, eine Prozedur WORT zu starten, die nicht definiert worden war. Das Programm INTERACT hat diesen Fehler entdeckt.

- WORT is undefined
  Sie haben versucht, eine Prozedur/Funktion WORT in einer neuen Definition anzuwenden, die nicht definiert worden war. Der Compiler hat diesen Fehler entdeckt. Diese Fehlermeldung führt zum Abbruch des Kompiliervorganges.

- Dictionary full
  Der Kodebereich und die Namensliste, die während des Betriebes aufeinander zu wachsen, sind kollidiert. Mit anderen Wort ist das Programm zu lang. Sie können dies dadurch umgehen, indem sie stärker modularisieren und die Programme nacheinander compilieren.

Notizen:

3. Die höhere Programmiersprache RPNL

3.1 Die Daten

RPNL verarbeitet 2 Arten von Daten, nämlich

- numerische Daten (Zahlen) und
- Zeichenketten (Strings).

Innerhalb der numerischen Daten werden 3 verschiedene Datentypen unterschieden, die ebenso wie die Zeichenketten im folgenden näher vorgestellt werden.

3.1.1 Die Datentypen

INTEGER

sind einmal Zahlen im mathematischen Sinne, für die darüber hinaus gilt

0 <= INTEGER <= 65535.

Also: RPNL verarbeitet natürliche Zahlen zur Basis 10, die nicht größer sind als 2**16-1.

Logische Werte (BOOLEAN)

Dieser Datentyp umfaßt die beiden Wahrheitswerte "wahr" und "falsch". Wenn oben behauptet wurde, dieser Typ gehöre zu den numerischen Daten, dann deshalb, weil die Wahrheitswerte durch Zahlen repräsentiert werden, nämlich

falsch - null
wahr - - 1 (0FFFFH).

Das Array

Der Datentyp Array faßt eine feste Anzahl von Elementen des gleichen oder auch unterschiedlichen Typs zusammen. Diese Datentypen können in RPNL alle Typen außer dem Array selbst sein. Bedingt durch den Aufbau der Sprache kann die Größe eines Array nur einmal fest vereinbart werden. Eine dynamische Veränderung, insbesondere aufgrund eingegebener Daten im Laufe eines Programmes ist nicht möglich. Eine weitere Einschränkung ist darin zu sehen, daß lediglich eindimensionale Arrays vereinbart werden können. Dieser Mangel kann aber durch geeignete Maßnahmen leicht umgangen werden. Beim Zugriff auf die Elemente eines Array wird übrigens deutlich, wie nahe RPNL an der Assemblerebene arbeitet, ohne dabei aber deren Nachteile in Kauf nehmen zu müssen (siehe hierzu die Beispielprogramme in Kapitel 4).

Der String

ist eine Zeichenkette, die aus einzelnen druckbaren Zeichen besteht. Wird innerhalb eines Programmes ein String vereinbart, so wird der zugehörige Text in Form einer Folge von Zeichen angegeben, die durch Hochkommata begrenzt sind. Wird die Zeichkette durch ein Eingabemedium eingelesen, so ist das zuerst eingegebene auch das erste Zeichen des String. Und das Zeichen vor dem Wagenrücklauf bzw Zeilenvorschub das letzte. Dies bedeutet insbesondere, daß diese Steuerzeichen nicht Bestandteil eines String sein können. Die Länge eines String ist auf maximal 255 Zeichen festgelegt. Das Format, das zur Speicherung eines String benutzt wird, sieht folgendermaßen aus:

Länge (1 Byte)
ASCII-Folge des Textes

Beispiel: 05 54 45 58 54
für 'TEXT'

3.1.2 Variablen und Konstanten

Gemeinsam ist Variablen und Konstanten, daß sie Platzhalter für Daten darstellen, denen vom Anwender Namen gegeben werden. Der Unterschied besteht darin, daß

Konstanten nach der Deklaration nicht veränderbar, also konstant sind, dagegen
Variablen ihren Wert im Laufe der Ausführung eines Programmes beliebig oft ändern können (natürlich vom Anwender gesteuert).

Vor der ersten Benutzung im Programm müssen alle verwendeten Namen (im obigen Sinne) deklariert werden (siehe hierzu auch Kap. 3.2.1). Dabei muß man sich entscheiden, ob dieser Platzhalter als Konstante oder Variable benutzt werden soll (dies trifft auf das Array nicht zu; Array-Elemente sind immer Variablen). Dabei hat man sich noch nicht festgelegt, für welche Datentypen diese Namen Verwendung finden sollen. Das geschieht im Prinzip erst dann, wenn sie (die Variable oder Konstante) zum ersten Male benutzt wird. Folgende Verwendungsmöglichkeiten ergeben sich für die Datentypen:

Datentyp	Verwendung möglich als
Integer	Variable oder Konstante
Boolean	Variable
Array	Variable
String	Variable oder Konstante

Da man das Diagramm auch umgekehrt lesen kann, wird die letzte Aussage bekräftigt (also kann beispielsweise eine Variable als Integer, Boolean oder String benutzt werden). Diese Aussage ist wichtig, ist aber möglicherweise erst in späterem Zusammenhang verständlicher.

3.1.3 Standardnamen

Einigen Konstanten sind schon Werte oder Bedeutungen von vorneherein zugeordnet und mit Standardnamen belegt; diese Namen sind für diese speziellen Zwecke reserviert. Sie haben eine stets gleichbleibende Bedeutung und müssen im Programm selbst durch Deklaration nicht erklärt werden. Folgende Standardnamen für Konstanten sind in RPNL festgelegt:

Name	Bedeutung	Datentyp
I	Laufindix der FOR-LOOP	Integer
TRUE	logisch "wahr"	Boolean
FALSE	logisch "falsch"	Boolean

Darüber hinaus sind Standardnamen vorbehalten für Prozeduren und Funktionen.

3.2 Aufbau der Programme

Jedes Programm besteht grundsätzlich aus drei Teilen, nämlich

- der Deklaration
- der Programmklammer mit Namen
- der Anweisungsliste.

Dabei können in Sonderfällen die Deklaration (wenn keine oder keine neuen Variablen benötigt werden) oder die Anweisungsliste (wenn nur Variablen oder Konstanten deklariert werden) fehlen. Zum Aufbau der Programme ist weiterhin zu beachten, daß alle aufeinanderfolgenden Schlüsselworte und Namen durch mindestens ein Leerzeichen (i.a. Space) getrennt werden müssen. Dort wo ein Leerzeichen stehen darf oder muß, dort dürfen auch mehrere stehen. Diese Regel gilt auch in ihrer Umkehrung. Diese Handhabung der Leerzeichen ist also die allgemein übliche.

3.2.1 Die Deklaration

Der Deklarationsteil wird eingeleitet durch das Schlüsselwort DECLARE und beendet mit DEND. Alle Namen von Programmen (klar!), aber auch von Variablen, Konstanten, Arrays oder Strings, allge-

mein also von Daten müssen vor ihrem ersten Aufruf deklariert, d.h. vereinbart werden.

Doch bevor Näheres zu der Deklaration der Datentypen gesagt wird, muß etwas zu den Namen ausgeführt werden. Namen sind in, RPNL beliebige Folgen von mindestens einem Zeichen, also Buchstaben, Ziffern oder Sonderzeichen. Die maximale Länge eines Namens darf 255 (!) Zeichen betragen. Die Nennung eines Namens innerhalb eines Programmes generiert eine Adresse, aber keinen Wert einer Variablen. Dies gilt lediglich für Konstanten. Dieser wesentliche Unterschied muß dringend beachtet werden, da sonst nichtlauffähige Programme erzeugt werden. Welche Bedeutung diese Adresse nun hat, kann einmal vom Programmierer bestimmt werden, oder sie ist bereits durch die Deklaration festgelegt. Es ist jedoch immer so, daß die Namen bzw die durch sie generierten Adressen auf Speicherstellen zeigen. Deshalb werden sie als einfache Zeiger benannt.

Bei der Deklaration erfolgen im wesentlichen zwei Dinge. Einmal wird ein Name als symbolische Adresse eingeführt (siehe oben), andererseits wird entsprechender Speicherraum reserviert. Es sind 3 Arten der Vereinbarung möglich:

VAR einfacher Zeiger

Es wird eine Variable mit der Länge von 16 Bit vereinbart. Sie ist durch den einfachen Zeiger ansprechbar. Ihr Anfangswert ist Null. Dieser Wert kann beliebig oft durch eine geeignete Anweisung geändert werden.

CONST einfacher Zeiger Zahl

Es wird eine Konstante mit der Länge 16 Bit vereinbart. Sie ist durch den einfachen Zeiger ansprechbar. Ihr Wert ist stets gleich der Zahl in der Vereinbarung. Dieser Wert kann im Programm nicht geändert werden.

ARRAY einfacher Zeiger Zahl

Es wird ein Speicherbereich von Variablen mit der Länge 16 Bit vereinbart. Und zwar genau so viele (Variablen) wie die Zahl angibt. Die Anfangswerte dieser Variablen sind allerdings nicht gleich Null, sondern zufällige Bitkombinationen. Wie Elemente von Arrays angesprochen werden, wird im Abschnitt 4 erklärt (Beispielprogramm "Sortieren eines Array).

STRING einfacher Zeiger 'Text'

weist dem Text Speicherplätze zu, auf die der Zeiger deutet. Die Ausgabe des Textes erfolgt mit PRINT(S).

Da nach einer Deklaration über die Namensliste von allen Programmteilen aus auf alle Variablen und Konstanten zugegriffen werden kann, sind solche deklarierten Variablen als global zu betrachten. Werden dagegen die repräsentierten Werte in irgendeiner Form aufgerufen und auf den Stack gelegt (beispielsweise als Parameterübergabe), so kann man nur lokal daraufzugreifen.

3.2.2 Die Programmklammer mit Namen

Ausnahmslos beginnt jedes Programm mit dem Schlüsselwort PROGRAM und endet mit END. Der Name des Programmes folgt dem Wort PROGRAM und dient der späteren Identifizierung. Identifizierung sollen heißen Ansprechbarkeit. Denn damit erst wird die Möglichkeit geschaffen, dieses Programm (-stück) ausführen zu lassen. Sei es als Unterprogramm, Hauptprogramm oder rekursiver Aufruf (dazu später mehr).

3.2.3 Die Anweisung

Dieser Programmteil enthält die Algorithmen, die die Lösung des gestellten Problems vollziehen. Algorithmen bestehen aus Anweisungen, die in einer bestimmten Reihenfolge, manchmal auch mehrmals auszuführen sind. Dazu stellt RPNL eine gewisse Anzahl von Schlüsselworten und Standardnamen zur Verfügung. Diese Menge läßt sich zweckmäßigerweise in 3 Gruppen einteilen:

- einfache Anweisungen
- Ein- und Ausgabeanweisungen
- Anweisungen zur Steuerung der Ausführung.

Bevor auf diese Gruppen im einzelnen näher eingegangen wird, muß der Begriff Operand, so wie er auch meistens verwendet wird, erklärt werden. Als Operand sind möglich:

Zahl

repräsentiert eine Folge von Ziffern, die einen Wert zwischen 0 und 65535 als Dezimalzahl repräsentieren.

Beispiele: 13 oder 4567 oder 49876
aber nicht -17
und nicht 89012.

Hier wird leider eine Einschränkung offensichtlich. Es werden nur Zahlen verarbeitet, die positiv und kleiner oder gleich der Zahl 65535 sind, also insbesondere keine negativen Zahlen. Allerdings wird man mit ein wenig RPNL-Praxis schnell merken, daß man einfache Routinen schreiben kann, die diese Mängel vollends beheben. Dagegen machen sich die relativ einfachen Routinen der Integerarithmetik in kurzen Ausführungszeiten vorteilhaft bemerkbar.

Einfacher Zeiger

Im Abschnitt 3.2.1 wurde dieser Begriff eingeführt und auch schon erklärt. Deshalb nur noch kurz etwas zu deren Aufbau. Neben ihrer syntaktischen Aufgabe tragen die einfachen Zeiger wesentlich zur Programmdokumentation bei. Denn durch eine geeignete Namensgebung steigt die Lesbarkeit der Programme. Deshalb sind dem Aufbau der einfachen Zeiger kaum Grenzen gesetzt, insbesondere nicht, was deren Länge anbelangt. Es ist aber darauf zu achten, daß keine Kollision mit den reservierten Standardnamen und Schlüsselworten auftritt.

Gültige Beispiele für einfache Zeiger sind:

WERT ZAHL NUMMER1 X+2.

Ausdruck

ist der verallgemeinerte Begriff für verarbeitete Daten (im Sinne des Kapitels 3, auf die die folgenden Prädikate angewandt wurden). Insbesondere kann ein Ausdruck wieder aus Ausdrücken zusammengesetzt sein, oder aber kann er einen Zeiger repräsentieren (was für den Zugriff auf ein Array wichtig ist). Dies zeigt ein wenig die Leistungsfähigkeit von RPNL.

3.2.3.1 Einfache Anweisungen

Es soll an dieser Stelle kurz auf die Besonderheiten der UPN hingewiesen werden. Der an sich einzige Unterschied zur arithmetischen (Infix-) Schreibweise, der aber anfangs zu Schwierigkeiten führen kann, besteht in der Reihenfolge der Operanden und Operatoren. Gewohnt ist man die Folge

Ergebnis := Operand 1 Operator Operand 2

oder bei einstelligen Prädikaten

Ergebnis := Operator Operand

Beispiele:

WERT := 37 + 14

oder

WURZEL := SQRT (36).

Das gleiche sieht bei der UPN folgendermaßen aus:

Operand 1 Operand 2 Operator (=) Ergebnis.

Das Gleichheitszeichen ist hier an sich nicht notwendig, da die

Anwendung des Operators automatisch das Ergebnis erzeugt. Und bei einstelligen Prädikaten

Operand Operator (=) Ergebnis.

Die gleichen Beispiele hierzu:

37 14 + (=) WERT

oder

36 SQRT (=) WURZEL.

Diese Schreibweise wird konsequent bei allen Anweisungen benutzt.

Beispiele:

- einfacher Ausdruck

  Datum1 Datum2 Operator1

- zusammengesetzter Ausdruck

  WertA WertB Operator1 WertC Operator2

  oder auch

  A B Operator1 Operator2 C Operator3

Es fällt auf, daß die zusammengesetzten Ausdrücke keine Art von Klammern aufweisen. Dies ist auch unnötig, da die Regelung der Prioritäten, wie sie aus der Mathematik bekannt ist (beispielsweise: "Punkt-geht vor Strichrechnung"), sich alleine aus der Reihenfolge der Nennung der Operanden und Operatoren ergibt. Damit werden auch aufwendige Auswertungsalgorithmen überflüssig.

Mit diesem Vorwissen können nun die einfachen Anweisungen vorgestellt werden. Diese Anweisungsart stellt letzten Endes immer einen Ausdruck dar, der einem Zeiger zugewiesen werden kann. Ganz salopp gesprochen geschieht hier folgendes. Auf ein oder zwei Operanden werden ein - oder zweistellige Operatoren angewendet. Das entstandene Ergebnis kann dann beliebig wieder mit anderen Operanden verknüpft werden. Oftmals wird es einem Zeiger zugewiesen werden, um es für eine spätere Verwendung abgespeichert und gesichert zu haben.

Welches sind nun ein- und zweistellige Operatoren ?

Als einstellige sind implementiert:

INC, DEC, NOT, DUP, "?".

Und als zweistellige Operatoren:

+, -, DIV, *, MOD, SWAP, AND, OR und die Vergleichsoperatoren.

Die entsprechende Syntax ergibt sich exakt aus dem Syntaxdiagramm. Anwendungsmöglichkeiten werden eingehend im Kapitel 4 durch Beispiele illustriert. Im folgenden wird die Semantik der Operatoren beschrieben. Zunächst die einstelligen (das sind also die Funktionen, die einen Operanden erwarten und verarbeiten).

Die einstelligen Operatoren

- -

INC

Aufruf: 1 Operand
Ergebnis: 1 Operand

inkrementiert einen aufgerufenen Wert, d.h. der Wert wird um 1 erhöht.

- -

- -

DEC

Aufruf: 1 Operand
Ergebnis: 1 Operand

dekrementiert einen aufgerufenen Wert, d.h. der Wert wird um 1 erniedrigt.

- -

NOT

Aufruf: 1 Operand
Ergebnis: 1 Operand

ist nur auf logische Werte anwendbar, also ein logischer Operator, hier der Negation.
Das Ergebnis ist das in der Logik Übliche (also wieder ein logischer Wert) und in der folgenden Wertetabelle angegeben:

Operand	Ergebnis
TRUE	FALSE
FALSE	TRUE

- -

DUP

Aufruf: 1 Operand
Ergebnis: 2 Operanden

dupliziert (nicht verdoppelt) einen aufgerufenen Wert. Der bei TOP präsente Wert wird auf TOP-1 kopiert, so daß bei TOP und TOP-1 identische Werte vorhanden sind.

- -

?(B) Aufruf: 1 Operand
Ergebnis: 1 Operand

Dieser Operation muß ein Zeiger vorangehen. Bei Aufruf wird dann der Inhalt der durch den Zeiger repräsentierten Speicherstelle (1 Byte) abgefragt und auf den Stack gelegt.

- -

? Aufruf: 1 Operand
Ergebnis: 1 Operand

Dieser Operation muß ein Zeiger vorangehen. Bei Aufruf wird der Inhalt der durch den Zeiger repräsentierten Speicherstelle (hier ist es 1 Doppelbyte bzw eine Adresse) abgefragt und auf den Stack gelegt (Umkehrung der Zuweisung).

- -

RND Aufruf: 0 Operanden
Ergebnis: 1 Operand

Bei Aufruf dieser Funktion wird eine Zufallszahl im Rahmen der Definition von Zahlen erzeugt und auf dem Stack abgelegt. Bei jedem ersten Aufruf der Funktion nach Starten des Compilers wird die gleiche Folge von Zufallszahlen erzeugt.

- -
- -

Die zweistelligen Operatoren

- -

:= Aufruf: 2 Operanden
Ergebnis: 0 Operanden

Diese Funktion stellt die Zuweisung dar. Sie erfordert als Operanden mindestens einen Zeiger, der dann als zweiter (in der Reihenfolge der Nennung) Operand auftreten muß. Diesem wird dann der Wert des ersten zugewiesen. Und zwar hier als Doppelbyte bzw als ein Zeiger oder eine Adresse.

- -

:=(B) Aufruf: 2 Operanden
Ergebnis: 0 Operanden

Auch diese Funktion stellt eine Zuweisung dar. Im Unterschied zu := wird hierbei der Wert des ersten Operanden in einem Byte abgelegt. Er ist also auf einen Wert <= 255 beschränkt.

- -

+ Aufruf: 2 Operanden
Ergebnis: 1 Operand

Addition von 2 Operanden.

- -

- Aufruf: 2 Operanden
Ergebnis: 1 Operand

Subtraktion von 2 Operanden

- -

DIV Aufruf: 2 Operanden
Ergebnis: 1 Operand

Ganzzahlige Division; der errechnete Rest ist nicht verfügbar (siehe hierzu MOD).

- -

* Aufruf: 2 Operanden
Ergebnis: 1 Operand

Multiplikation von 2 Operanden

- -

MOD Aufruf: 2 Operanden
Ergebnis: 1 Operand

Modulo-Division; es wird eine ganzzahlige Division durchgeführt, wovon aber nur der erechnete Rest verfügbar ist.

- -

SWAP Aufruf: 2 Operanden
Ergebnis: 2 Operanden

Vertauschen von Operand 1 und Operand 2 (bzw. es werden TOP und TOP-1 vertauscht, siehe hierzu auch Kapitel 2); die Operanden selbst werden nicht geändert.

- -

AND Aufruf: 2 Operanden
Ergebnis: 1 Operand

ist nur auf logische Werte anwendbar, also ein logischer Operator, hier der Konjunktion. Das Ergebnis ist das in der Logik übliche (also wieder ein logischer Wert) und in der folgenden Wahrheitstabelle angegeben:

Operand 1	Operand 2	Ergebnis
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

OR Aufruf: 2 Operanden
Ergebnis: 1 Operand

ist nur auf logische Werte anwendbar, also ein logischer Operator, hier der Disjunktion. Das Ergebnis ist das in der Logik übliche (also wieder ein logischer Wert) und in der folgenden Wahrheitstabelle festgehalten:

Operand 1	Operand 2	Ergebnis
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- -

Vergleichsoperatoren Aufruf: 2 Operanden
Ergebnis: 1 Operand

liefern bei Anwendung auf Operanden, für die eine Ordnung definiert ist (das ist insbesondere der Typ INTEGER), logische Werte als Ergebnis. Allerdings lassen sich nicht alle Vergleichsoperatoren auf alle Datentypen anwenden, d.h. Datentypen können mit einigen Vergleichsoperatoren unverträglich sein oder auch nicht sinnvoll. Deshalb sind die anwendbaren Datentypen in der folgenden Tabelle mit vermerkt.

Operator	Bedeutung	anwendbar auf
=	ist gleich	Integer, Boolean, Array
<>	ist ungleich	Integer, Boolean, Array
<	ist kleiner als	Integer, Array
>	ist größer als	Integer, Array
<=	ist nicht größer als ist kleiner oder gleich	Integer, Array
>=	ist nicht kleiner als ist größer oder gleich	Integer, Array
=(S)	ist gleich	String
>=(S)	ist nicht kleiner als ist größer oder gleich	String

Es gibt noch einen weiteren Vergleichsoperator, der in der Tabelle nicht aufgeführt ist, weil ihm eine Besonderheit eigen ist. Es ist dies der Operator 0= . Er erwartet und verarbeitet nur einen Operanden und vergleicht diesen mit Null. Das Ergebnis ist der entsprechende logische Wert.

3.2.3.2 Ein erstes Programmbeispiel

Mit dem bisher erworbenen Wissen soll nun ein erstes Programm in der Programmiersprache RPNL formuliert werden. Dabei wird langsam vorgegangen und alle notwendigen Schritte eingehend erläutert. Wenn an einigen Stellen Anweisungen benutzt werden, die regulär erst in einem späteren Abschnitt eingeführt werden, dann ist deren Gebrauch so eindeutig, daß keine Mißverständnisse auftreten können.

Wurzelziehen

Aufgabe:

Eine Funktion WURZEL zieht die Quadratwurzel aus einer natürlichen Zahl. Der Radikant ist als TOP auf dem Stack abgelegt, das Ergebnis, die Quadratwurzel, ist nach dem Aufruf WURZEL wieder als TOP auf dem Stack. Mit einem Programm SQRT soll die Funktion WURZEL aufgerufen und getestet werden.

Algorithmus:

Es wird die Iterationsvorschrift

$$X_{n+1} := \frac{X_n + Y / X_n}{2} \qquad \text{mit } X_n := Y$$

benutzt. Auf deren Herleitung und Terminierung soll an dieser Stelle nicht eingegangen werden. Entscheidend ist nur, ein geeignetes Abbruchkriterium festzulegen. Naheliegend ist es, mit der Iteration dann aufzuhören, wenn sie keine Verbesserung mehr bringt, also

$$X_n = X_{n+1}$$

Dieses Kriterium ist zumindest dann sinnvoll, wenn der Datentyp INTEGER verwendet wird. Als Startwert für X wählen wir Y selbst.

Das Programm

Zunächst soll wie gefordert die Funktion WURZEL definiert werden. Dazu wird die Iterationsvorschrift umgangssprachlich formuliert (programmiert):

```
PROGRAMM  WURZEL
     DEKLARATION der Variablen
     ÜBERNAHME der Argmente (Y)
     WIEDERHOLE folgendes
          X      :=  ( X  + Y div X  ) div 2,
           n+1          n           n

     BIS  X     = X
           n+1     1

     ENDE der Schleife
     ÜBERGABE der Argumente (X    ),
                              n+1
ENDE
```

Der nächste Schritt in der Programmentwicklung ist die Festlegung der Namen für die Variablen.

Es sei:

Y	---	Y
X_n	---	XALT
X_{n+1}	---	XNEU

Damit kann die Deklaration geschrieben werden als

```
DECLARE
        VAR Y   VAR XALT   VAR XNEU
DEND.
```

Der Radikant, also Y, wird als TOP übergeben. Da dieser Wert aber mehrmals gebraucht wird, muß TOP der Variablen Y zugewiesen und damit zwischengespeichert werden. Ebenso muß der Variablen XALT der Startwert Y übertragen werden. Der Speicherplatz TOP wird also zweimal gebraucht; deshalb wird er vor der ersten Zuweisung, die ihn zerstören würde, dupliziert. Demnach sieht die Übernahme der Argumente so aus:

```
DUP   Y   :=        XALT    :=     .
```

Nun gilt es, die Iterationsvorschrift, die in der Infixnotation vorliegt, in die UPN bzw in RPNL umzuformen. Dazu benutzen wir einen Baum, der folgendermaßen aufgebaut ist: Jeder Ast zeigt auf einen Operanden und jeder Knoten stellt eine Operation dar. An der Wurzel steht das Ergebnis. Es fällt intuitiv nicht schwer, unsere Iterationsvorschrift darzustellen als:

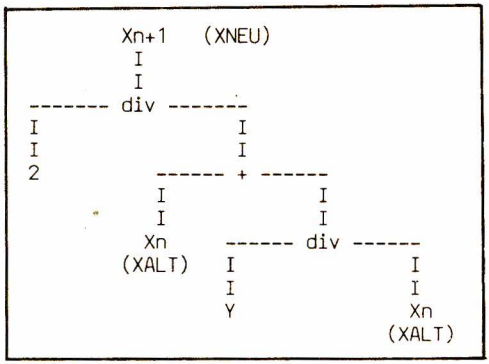


Bild 12 **Die Erzeugung der UPN - Form durch eine Baumstruktur**

Durch den Rückgriff auf den Baum wird offensichtlich, in welcher Reihenfolge die einzelnen Operationen abzuarbeiten sind (da ja keine expliziten Klammern zur Verfügung stehen. In Wirklichkeit kann natürlich bis zur Tiefe des Stack geklammert werden.). Also wird die Iteration in RPNL folgendermaßen definiert:

```
Y     ?     XALT  ?     DIV
XALT  ?  +  2           DIV         XNEU  :=
```

Die Schleifenabbruchbedingung ergibt sich klar als

```
XALT   ?      XNEU   ?      =
```

und die Umspeicherung

```
XNEU   ?      XALT   :=      .
```

Nach der Schleife ist lediglich noch die Übergabe der Argumente zu formulieren. Gemäß der Konvention in der Aufgabenstellung legen wir mit

```
XNEU   ?
```

das Ergebnis als TOP auf den Stack. Nachdem nun alle Schritte einzeln formuliert sind, können wir diese zu unserem RPNL-Programm zusammensetzen:

```
DECLARE
        VAR Y VAR XALT VAR XNEU
        VAR ZAHL          ; WIRD ERST SPÄTER GEBRAUCHT
DEND

PROGRAM WURZEL
        DUP  Y :=  XALT :=
        REPEAT
                Y ?  XALT ?  DIV
                XALT ?  + 2 DIV XNEU :=
```

```
                XALT ? XNEU ? = UNTIL
                XNEU ? XALT :=
                LOOP
                XNEU ?
        END
```

An dieser Stelle sollen einmal zur Veranschaulichung die Bewegungen auf dem Stack während der Abarbeitung dargestellt werden. Es ist empfehlenswert, beim Einstieg in RPNL dieses Schema bei den ersten Programmentwürfen zu benutzen, da man sonst allzu schnell Schiffbruch erleidet.

Anweisungs-schritt	Stack TOP	X=don't care TOP-1	TOP-2
VAR Y	RADIKANT	X	X
VAR XALT	RADIKANT	X	X
VAR XNEU	RADIKANT	X	X
DUP	RADIKANT	RADIKANT	X
Y	ADRESSE Y	RADIKANT	RADIKANT
:=	RADIKANT	X	X
XALT	ADRESSE XALT	RADIKANT	X
:=	X	X	X
REPEAT	X	X	X
Y ? XALT ?	(XALT)	(Y)	X
DIV	(Y) DIV (XALT)	X	X
XALT ?	(XALT)	(Y)DIV(XALT)	X
+	(XALT)+(Y)DIV(XALT)	X	X
2 DIV	((XALT)+	X	X
	(Y)DIV(XALT))DIV 2	X	X
XNEU :=	X	X	X
XALT ?	(XALT)	X	X
XNEU ?	(XNEU)	(XALT)	X
=	TRUE/FALSE	X	X
UNTIL	X	X	X
XNEU ?	(XNEU)	X	X
XALT :=	X	X	X
LOOP	X	X	X
XNEU ?	(XNEU)	X	X

Um die Funktion WURZEL zu testen, wird anschließend ein Programm SQRT entwickelt. SQRT hat allerdings weniger die Aufgabe, WURZEL auf seine Richtigkeit zu überprüfen, als vielmehr nur eine Zahl einzulesen, den Argumentwert zu übergeben, WURZEL aufzurufen und das Ergebnis anzuzeigen. Auf eine umgangssprachliche Formulierung der Aufgabe wird verzichtet. Das Programm SQRT lautet in RPNL:

```
        PROGRAM SQRT
                ZAHL READ
                ZAHL ? WURZEL ZAHL :=
                CR
                ZAHL PRINT
        END
```

Wir haben nun zwei lauffähige Programme entwickelt, die wir in den Computer eingeben wollen. Wir wissen, daß der Rechner dazu im Status INTERACT stehen muß. Dies ist kein Problem, denn das ist er nach dem Start des Compilers und nach Ausführung eines Programmes automatisch. Wir wollen uns vielmehr überlegen, in welcher Reihenfolge wir SQRT und WURZEL eingeben müssen. Ist eine bestimmte Reihenfolge zwingend oder ist sie beliebig?

Wenn wir uns an den Abschnitt erinnern, in dem der Aufbau der Namensliste beschrieben ist, so wissen wir, daß Namen vom Compiler nur dann akzeptiert werden, wenn sie bereits deklariert worden sind.

Schauen wir uns die beiden Programme diesbezüglich an, so stellen wir fest, daß ausschließlich folgende Reihenfolge der Eingabe möglich ist: WURZEL und dann SQRT. Die Begründug hierfür ist jetzt naheliegend. Geben wir nämlich zunächst SQRT ein, so erhalten wir bei Prüfung des darin vorhandenen Namens WURZEL eine Fehlermeldung (WURZEL ist ja noch nicht definiert bzw in die Namensliste eingetragen).

Wenn Programme stark strukturiert sind und in viele Module gegliedert, ist dieses Problem oft gravierend und nur eine korrekte Planung des Programmes ist hier eine Hilfe.

3.2.3.3 Die Ein- und Ausgabeanweisungen

Während die einfachen Anweisungen die Daten manipulieren und vergleichen, stellt die Gruppe der Ein- und Ausgabeanweisungen die Verbindung des Rechners zur Außenwelt dar. RPNL stellt dafür eine Reihe von Prozeduren zur Verfügung, die es einmal gestatten, Werte (Daten) einzulesen oder aber auch Werte (Daten) auszugeben. Es wurde absichtlich vermieden, sich auf irgendwelche Ein- oder Ausgabemedien festzulegen. Es kommen zwar meistens nur die Tastatur, Bildschirm oder Drucker in Frage, dies muß aber nicht so sein. Im folgenden wird die Semantik der einzelnen Prozeduren erklärt; die Syntax ergibt sich aus den Überschriften.

CR

bewirkt auf dem Ausgabemedium einen Wagenrücklauf und einen Zeilenvorschub, d.h. der Cursor wird an den Anfang der nächsten Zeile geschoben.

WRITE 'Text'

gibt bei Aufruf den Text aus, d.h. die Zeichenfolge, die durch Hochkommata eingeschlossen dem Standardnamen WRITE folgt. Der Text darf das Zeilenende nicht überschreiten. Die Zeilenlänge ist auf 80 Zeichen eingestellt.

Beispiel: WRITE 'RPNL-Compiler'

Zeiger PRINT

stellt auf dem Ausgabemedium eine Integer-Zahl dar und zwar genau die, auf die der vorangestellte Zeiger deutet. Die Zahl wird sechsstellig ohne Vorzeichen dargestellt; führende Nullen werden unterdrückt.

Beispiel: WERT PRINT

Zeiger PRINT(S)

gibt eine als String zugewiesene Zeichenfolge aus und zwar genau die, auf die der zugehörige Zeiger deutet. Die Zeichenfolge muß vor der ersten Ausgabe deklariert (siehe hierzu unter STRING) oder eingelesen (siehe hierzu unter READ(S)) worden sein.

Beispiel:

```
TEXT STRING 'Dies ist ein Beispiel'
        :
        :
TEXT PRINT(S)
```

OUTPUT

Diese Prozedur gibt ein Zeichen auf der Position auf dem Bildschirm aus, an der der Cursor steht. Das auszugebende Zeichen muß im Register C stehen. Die Prozedur wird als Unterprogramm vom Programmtyp 1 oder 2 augerufen.

Beispiel:

```
CODE AUSGABE
; DIESES PROGRAMM GIBT 16 BLANKS AUS
;
        06 10          ; LD    B,10H
        0E 20          ; LD    C,20H
        CD OUTPUT      ; CALL OUTPUT
        10 FB          ; DJNZ OUTPUT
CEND
```

Wert OUTCHAR

Auch diese Prozedur gibt genau ein Zeichen aus, nämlich das an oberster Stackposition (also Wert).

Beispiel:

```
65 CHAR :=
CHAR ? OUTCHAR  ; GIBT EIN "A" AUS
```

oder

```
65 OUTCHAR        ; GIBT AUCH EIN "A" AUS
```

Zeiger READ

liest eine Integer-Zahl von einem Eingabemedium und weist sie dem Speicherplatz zu, auf den der Zeiger deutet.

Beispiel: WERT READ

Zeiger READ(S)

liest eine Zeichenfolge ein und weist dieser Speicherplätze zu, auf die durch Zeiger zugegriffen werden kann. Die Eingabe wird durch Wagenrücklauf bzw. Zeilenvorschub beendet. Die erzeugte Zeichenkette wird auf einer Adresse abgelegt, die vor der Eingabe unter der Variablen SYSFIELD abfragbar ist. Nach Beendigung der Eingabe zeigt SYSFIELD auf die nächste freie Speicherstelle, an der Text abgelegt werden kann.

Für ein Beispiel wollen wir einen Artikelnamen gefolgt von der zugehörigen Anzahl einlesen und wieder ausgeben lassen. Zur Eingabe der Daten wird mit Kommentaren aufgefordert. Das RPNL Programm zur Lösung der Aufgabe sieht so aus:

```
DECLARE
     VAR ANZAHL      VAR NAME
DEND

PROGRAM ARTIKELANZAHL
     WRITE 'ARTIKELNAME:    '
     NAME READ(S)
     WRITE 'BESTAND:        '
     ANZAHL READ    CR CR
     WRITE '   ++  AUSGABE  ++'    CR
     ANZAHL PRINT        WRITE '      '
     NAME PRINT(S)
END
```

Das Programm erzeugt folgenden Ausdruck:

```
ARTIKELNAME:   CPU-Karte
BESTAND:       17


  ++  AUSGABE  ++
   17     CPU-Karte
```

Zeiger READ(CH)

Diese Prozedur hat die gleiche Funktion wie READ(S) mit dem Unterschied, daß eingegebene Kleinbuchstaben zu Großbuchstaben konvertiert werden.

INCHAR

Diese Aktion liest genau ein Zeichen von der Tastatur ein und legt es unter der Variablen ZEICHEN ab.

3.2.3.4 Anweisungen zur Steuerung der Anweisungen

Mit den Anweisungen, die bislang eingeführt wurden, ist es möglich, vollständige Programme zu schreiben. Allerdings sind dies lediglich lineare Abläufe, auf deren Verlauf nicht durch Variation der Daten Einfluß genommen werden kann. Genau dazu aber befähigt diese Gruppe von Anweisungen. Es wird durch sie die Möglichkeit eröffnet, abhängig von der Erfüllung einer Bedingung Anweisungen ausführen zu lassen oder nicht. RPNL unterstützt damit eine strukturierte Programmierung. An die Stelle von unübersichtlichen GOTO-Statements treten REPEAT-UNTIL-LOOP, IF-ELSE-THEN Bedingungsanweisungen und andere Schleifenbedingungen.

IF-ELSE-THEN

Vor dem Schlüsselwort IF steht ein Ausdruck, der die Bedingung (also einen logischen Wert) darstellt und dahinter eine Anweisungsliste, die nur dann ausgeführt wird, wenn die Bedingung erfüllt ist (den logischen Wert TRUE hat). Abgeschlossen wird die Liste durch das Schlüsselwort THEN. Will man jemanden zur Eingabe einer ungeraden Zahl auffordern und dies auch überprüfen, so kann man die Aufgabe folgendermaßen angehen:

```
DECLARE
        VAR  ZAHL
DEND

PROGRAM  UNGERADE
        WRITE 'Geben Sie eine UNGERADE Zahl ein'
        CR  ZAHL  READ
        ZAHL  ?  2  MOD 1 = IF
                WRITE 'Dies ist eine ungerade Zahl'
        THEN
END
```

Die Forderung, daß man auf den Fall der Nichterfüllung der Bedingung gesondert zugreifen kann, ist naheliegend. Dazu kann im Bedarfsfalle das Schlüsselwort ELSE vor THEN eingefügt werden. Die Anweisungsliste, die hinter dem ELSE steht, wird dann ausgeführt, wenn die Bedingung vor dem IF nicht zutrifft (den logischen Wert FALSE hat). So kann die Eingabe einer geraden Zahl in

obigem Beispiel folgendermaßen reklamiert werden.

```
DECLARE
        VAR  ZAHL
DEND

PROGRAM  UNGERADE
        WRITE 'Geben Sie eine UNGERADE Zahl ein'
        CR  ZAHL  READ
        ZAHL  ?  2  MOD 1 = IF
                WRITE 'Dies ist eine ungerade  Zahl'
        ELSE
                WRITE 'FEHLER! Dies ist eine gerade Zahl!'
        THEN
END
```

An dieser Stelle soll noch einmal an das Beispiel angeknüpft werden, das die Quadratwurzeln aus INTEGER-Zahlen berechnet hat. Man kann sich nun überlegen, ob für die Funktion WURZEL überhaupt alle Argumentwerte zugelassen sind. Nun, negative sicher nicht. Die sind für uns aber ungefährlich, weil Integerzahlen definitionsgemäß nichtnegative, ganze Zahlen sind. Aber wie steht es mit der Null ? Gemäß Vereinbarung gilt beim ersten Iterationsschritt Y = XALT. Im betrachteten Fall also gleich Null, was zwangsläufig schon beim ersten Rechenschritt zu dem unbestimmten Ausdruck 0/0 führt. Also muß die Null zwar nicht als Argument ausgeschlossen werden, denn da ist sie zugelassen, aber als Wert bei der Iteration. Dies geschieht durch eine zusätzliche IF-ELSE-THEN Bedingung. Die vollständig korrekte Funktion WURZEL hat dann folgendes Aussehen:

```
DECLARE
        VAR Y   VAR XALT        VAR XNEU
DEND

PROGRAM WURZEL
        DUP
        0= IF
        ELSE
                DUP  Y :=  XALT :=
                REPEAT
                        Y ?  XALT ?  DIV
                        XALT ?  + 2 DIV XNEU :=
                XALT ? XNEU ? = UNTIL
                        XNEU ? XALT :=
                LOOP
                XNEU ?
        THEN
END
```

Das führt zur allgemeinen Struktur von IF-ELSE-THEN Bedingungsanweisungen. Insbesondere können diese Anweisungen dann geschachtelt auftreten, wenn mehr als eine Alternative zur Auswahl steht. Dies wird dann in der folgenden Form realisiert:

```
Bedingung1      IF
     Anweisungsliste
                ELSE
                Bedingung2      IF
                     Anweisungsliste
                                ELSE
                                Bedingung3      IF
                                     Anweisungsliste
                                                ELSE
                                                .
                                                    .
                                                      .
                                                THEN
                                THEN
                THEN
```

Es muß hierbei darauf geachtet geachtet werden, daß jedes IF durch ein korrespondierendes THEN abgeschlossen wird. Als Beispiel betrachten wird das folgende Zahlenspiel, das von den verschachtelten Bedingungsanweisungen Gebrauch macht.

Ein Zahlenspiel

Dieses Spiel, daß auf einer Idee von Martin Gardner beruht, kann sich nicht auf bedeutende statistische oder andere mathematische Erkenntnisse berufen. Die Entscheidungen des Computers sind rein zufällig. Aber das gerade macht vielleicht den Reiz des Spieles aus.

Das Programm

Der Algorithmus ist so einfach, daß nach den bisherigen Kenntnissen der Programmiersprache RPNL dieses Programm auch ohne weitere Erklärungen verständlich ist.

```
DECLARE
        STRING JA 'JA'
        VAR DIFF        VAR ANTWORT     VAR SPIELER
        VAR COMP        VAR ANZAHL      VAR CZAHL
        VAR SZAHL
DEND


PROGRAM  ERKLAERUNGEN
    CR CR
    WRITE ' WIR DENKEN UNS BEIDE EINE POSITIVE ZAHL AUS.'
    WRITE ' DANN VERGLEICHEN WIR DIE BEIDEN ZAHLEN.' CR
    WRITE ' WER DIE KLEINERE HAT, GEWINNT EINEN PUNKT.' CR
    WRITE ' ALLERDINGS, WENN SIE GERADE UM 1 KLEINER IST,'
    WRITE ' DANN BEKOMMT DER ANDERE 2 PUNKTE!' CR CR
    WRITE ' EIN BEISPIEL:' CR
    WRITE ' WENN IHRE ZAHL 15 UND MEINE 20 IST, DANN' CR
    WRITE ' BEKOMMEN SIE EINEN PUNKT.' CR
    WRITE ' ABER WENN SIE 15 UND ICH 16 HABE, DANN KRIEGE ICH' CR
    WRITE ' STOLZE 2 PUNKTE UND SIE --- LEIDER NICHTS!' CR CR
```

```
    WRITE '    DAS HABEN SIE DOCH AUCH VERSTANDEN ?' CR CR
END

PROGRAM  INTEGER
      CR CR
      WRITE '    ZAHLEN-SPIEL'
      CR CR
      WRITE '     WOLLEN SIE ERKLAERUNGEN ?  '
      ANTWORT  READ(S)
      ANTWORT ?  JA ?  =(S)  IF
              ERKLAERUNGEN
      THEN
      0 SPIELER  :=     0  COMP  :=     0  ANZAHL  :=
      REPEAT
            RND  5  MOD  INC  CZAHL  :=
            CR  WRITE '     ICH HABE MEINE ZAHL'  CR
            WRITE '     WIE HEISST DENN IHRE ?     '
            SZAHL READ
            WRITE '            MEINE HEISST  :'
            CZAHL  PRINT
            CZAHL  ?  SZAHL  ?  -  DIFF  :=
            DIFF  ?  0=  IF
                  WRITE '     NA, DAS MUESSEN WIR'
                  WRITE ' NOCH EINMAL VERSUCHEN'
                  ANZAHL ?  DEC  ANZAHL  :=
            ELSE
                  DIFF ?  65535  =  IF
                        WRITE '   DAS SIND 2 PUNKTE FUER SIE!'
                        SPIELER  ?  2  +  SPIELER  :=
                  ELSE
                        DIFF  ?  65535 <
                        DIFF  ?  32767 >  AND  IF
                              WRITE '     DAS IST 1'
                              WRITE ' PUNKT FUER MICH'
                              COMP  ?  INC COMP  :=
                        ELSE
                              DIFF  ?  1  >  IF
                                    WRITE '    DAS IST 1'
                                    WRITE ' PUNKT FUER SIE'
                                    SPIELER  ?  INC
                                          SPIELER  :=
                              ELSE
                                    WRITE '     EU EU EU'
                                    WRITE ' DAS SIND 2 PUNKTE'
                                    WRITE ' FUER MICH!!!!'
                                    COMP  ?  2  +  COMP  :=
                              THEN
                        THEN
                  THEN
            THEN
            ANZAHL  ?  INC  ANZAHL  :=
            ANZAHL  ?  5  =  IF
                  CR  CR
                  WRITE '     DER SPIELSTAND IST JETZT'  CR
                  WRITE '     ========================'  CR
```

```
                    WRITE '     ICH :'   COMP  PRINT       CR
                    WRITE '     SIE :'   SPIELER  PRINT    CR
                    WRITE '     ========================'  CR
                    WRITE '     WOLLEN SIE WEITERMACHEN ?   '
                    ANTWORT READ(S)
                    0  ANZAHL  :=
            ELSE
                    JA  ?  ANTWORT  :=
            THEN
      ANTWORT  ?  JA  ?  =(S)  NOT  UNTIL
      LOOP
      CR CR
      COMP  ?  SPIELER  ?  >  IF
            WRITE '     MACHEN SIE SICH NICHTS DRAUS' CR
            WRITE '     DAS NAECHSTE MAL HABEN SIE MEHR GLUECK!'
            CR
      ELSE
            WRITE '     FEIGLING!!!!' CR
            WRITE '     AUFHOEREN, WENN SIE GERADE MAL NICHT'
            WRITE ' VERLIEREN;' CR
            WRITE '     DAS IST TYPISCH.'
      THEN
      CR CR
END

Ok!
#INTEGER

     ZAHLEN-SPIEL

     WOLLEN SIE ERKLAERUNGEN ?  JA
     WIR DENKEN UNS BEIDE EINE POSITIVE ZAHL AUS.
     DANN VERGLEICHEN WIR DIE BEIDEN ZAHLEN.
     WER DIE KLEINERE HAT, GEWINNT EINEN PUNKT.
     ALLERDINGS, WENN SIE GERADE UM 1 KLEINER IST,
     DANN BEKOMMT DER ANDERE 2 PUNKTE!

     EIN BEISPIEL:
     WENN IHRE ZAHL 15 UND MEINE 20 IST, DANN
     BEKOMMEN SIE EINEN PUNKT.
     ABER WENN SIE 15 UND ICH 16 HABE, DANN KRIEGE ICH
     STOLZE 2 PUNKTE UND SIE --- LEIDER NICHTS!

     DAS HABEN SIE DOCH AUCH VERSTANDEN ?


     ICH HABE MEINE ZAHL
     WIE HEISST DENN IHRE ?     3
            MEINE HEISST  :     4     EU EU EU! DAS SIND 2 PUNKTE
                                                FUER MICH!!!!
     ICH HABE MEINE ZAHL
     WIE HEISST DENN IHRE ?     4
                              :
                              :
                              :
```

```
WIE HEISST DENN IHRE ?     3
       MEINE HEISST  :     1     DAS IST 1 PUNKT FUER MICH

DER SPIELSTAND IST JETZT
========================
ICH :     9
SIE :     7
========================
WOLLEN SIE WEITERMACHEN ?   JA

ICH HABE MEINE ZAHL
WIE HEISST DENN IHRE ?     4
       MEINE HEISST  :     4     NA, DAS MUESSEN WIR NOCH
                                        EINMAL VERSUCHEN
ICH HABE MEINE ZAHL
WIE HEISST DENN IHRE ?     4
       MEINE HEISST  :     5     EU EU EU! DAS SIND 2 PUNKTE
                                          FUER MICH!!!!
ICH HABE MEINE ZAHL
                         :
                         :
                         :
WIE HEISST DENN IHRE ?     1
       MEINE HEISST  :     5     DAS IST 1 PUNKT FUER SIE

DER SPIELSTAND IST JETZT
========================
ICH :     4
SIE :     5
========================
WOLLEN SIE WEITERMACHEN ?   JA

ICH HABE MEINE ZAHL
WIE HEISST DENN IHRE ?     5
       MEINE HEISST  :     1     DAS IST 1 PUNKT FUER MICH
ICH HABE MEINE ZAHL
                         :
                         :
                         :
DER SPIELSTAND IST JETZT
========================
ICH :    14
SIE :    10
========================
WOLLEN SIE WEITERMACHEN ?   NEIN

MACHEN SIE SICH NICHTS DRAUS
DAS NAECHSTE MAL HABEN SIE MEHR GLUECK!
Ok!
#
```

REPEAT-UNTIL-LOOP

Im Gegensatz zur Bedingungsanweisung wird bei den Wiederholungsanweisungen der Wert einer Bedingung dazu benutzt, eine Liste von Anweisungen wiederholt ausführen zu lassen oder deren Ausführung abzubrechen. Diese Bedingung nennen wir das Abbruchkriterium. Die Wiederholungsanweisungen werden auch als Schleifen bezeichnet. Ganz allgemein ist die Struktur einer solchen Anweisung folgendermaßen:

```
SCHLEIFENANFANG

        ANWEISUNGSLISTE 1

BEDINGUNG

        ANWEISUNGSLISTE 2

SCHLEIFENENDE
```

Diese Anweisung bewirkt folgendes: Beim ersten Eintritt in die Schleife wird die ANWEISUNGSLISTE 1 auf jeden Fall, d.h. unabhängig vom Wert der Bedingung ausgeführt. Anders bei der ANWEISUNGSLISTE 2. Denn ist der Wert der BEDINGUNG "wahr", dann wird diese Liste nicht ausgeführt, sondern sofort zum SCHLEIFENENDE gesprungen. Im anderen Fall, also wenn die BEDINGUNG "falsch" war, wird auch die ANWEISUNGSLISTE 2 ausgeführt und wieder zum SCHLEIFENANFANG gesprungen. In dieser Allgemeinheit wird die Wiederholungsanweisung von kaum einer Programmiersprache angeboten. In RPNL ist diese Schleife durch die REPEAT-UNTIL-LOOP realisiert.

Hinter dem Schlüsselwort REPEAT folgt eine Anweisungsliste, die alle die Anweisungen enthält, die wiederholt werden sollen. Abgeschlossen wird die Schleife durch das Schlüsselwort LOOP. An einem geeigneten Punkt zwischen REPEAT und LOOP wird das Schlüsselwort UNTIL geschoben, dem direkt ein Ausdruck (das ist die Bedingung) vorangeht. Abhängig vom aktuellen Wert des Ausdruckes erfolgt die Fortsetzung des Programmes mit der ersten Anweisung hinter UNTIL (wenn der Ausdruck den Wert FALSE hat) oder hinter dem Schlüsselwort LOOP (wenn der Ausdruck TRUE ergibt).

Die REPEAT-UNTIL-LOOP erinnert sehr an die Wiederholungsschleife in PASCAL. Wird in RPNL das Schlüsselwort direkt vor dem LOOP plaziert (wenn also die zweite Anweisungsliste leer ist), dann hat sie exakt die gleiche Bedeutung. Auch die aus PASCAL bekannte WHILE-DO-Schleife läßt sich so simulieren. Nämlich dann, wenn das UNTIL direkt hinter dem REPEAT folgt (wenn die erste Anweisungsliste leer ist). Allerdings muß dazu der Ausdruck vor dem UNTIL negiert werden (mit dem einstelligen Prädikat NOT). Um das weitere Verständnis zu erleichtern, soll zunächst ein einfaches Beispielprogramm eingestreut werden.

Primzahlfaktorisierung

Der Leser kann zur eigenen Übung am folgenden Beispiel selbst

herausfinden, wie die Struktur der beiden geschachtelten Schleifen aussieht. Insbesondere soll dargestellt werden, wo die o.a. Anweisungslisten beginnen bzw. enden.

```
DECLARE
        VAR TEILER   VAR ZAHL
DEND

PROGRAM   PRIMZAHL
        2   TEILER  :=
        WRITE '     Zahl ?     '
        ZAHL READ
        WRITE '     Primfaktoren von '
        ZAHL  PRINT
        REPEAT
        TEILER ?   TEILER ?  *  ZAHL ?  > UNTIL
                REPEAT
                ZAHL ?  TEILER ?  MOD 0= UNTIL
                        TEILER ?  2  =  IF
                                3  TEILER  :=
                                        ELSE
                                TEILER  ?  2  +  TEILER  :=
                                        THEN
                LOOP
                ZAHL ?  TEILER ?  DIV  ZAHL  :=
                CR  TEILER PRINT
        LOOP
        ZAHL ?  1 >=  IF
                CR    ZAHL  PRINT
                      THEN
END


        Ok!
        #PRIMZAHL
             Zahl ?     60060
             Primfaktoren von  60060
             2
             2
             3
             5
             7
            11
            13

        #Ok!
```

Im Beispiel wurde die Möglichkeit, Schleifen zu verschachteln schon angewandt. Für den allgemeinen Fall soll deren Gestalt hier noch einmal skizziert werden:

```
        REPEAT
             ANWEISUNGSLISTE 1/1
             REPEAT
                  ANWEISUNGSLISTE 2/1
```

```
            BEDINGUNG UNTIL
                 ANWEISUNGSLISTE 2/2
            LOOP
            ANWEISUNGSLISTE 1/2
        BEDINGUNG UNTIL
            ANWEISUNGSLISTE 1/3
            REPEAT
                 ANWEISUNGSLISTE 3/1
            BEDINGUNG UNTIL
                 ANWEISUNGSLISTE 3/2
            LOOP
            ANWEISUNGSLISTE 1/4
        LOOP
```

Es muß noch daraufhingewiesen werden, daß sämtliche Anweisungslisten entweder leer sein oder aber auch weitere Schleifen enthalten können.

FOR-LOOP

Die REPEAT-UNTIL-LOOP wird meistens angewandt, wenn die Anzahl der Schleifendurchläufe vor Ausführungsbeginn nicht bekannt ist und von Berechnungen innerhalb der Schleife abhängig ist. Im Gegensatz dazu steht bei der FOR-LOOP die Iterationshäufigkeit vor dem ersten Durchlauf fest.

Dem Schlüsselwort FOR gehen zwei Operanden (Laufvariablen) voraus, wovon der erste dem Endwert und der zweite dem Startwert entspricht. Zwischen dem FOR und dem begrenzenden Wort LOOP steht die Anweisungsliste, die genausooft ausgeführt wird, wie die Differenz von End- und Anfangswert angibt. Dabei ist zu beachten, daß für den Anfangs- und Endwert nur positive Werte und lediglich die Schrittweite 1 möglich sind. Verschachtelte FOR-LOOPs sind durchaus möglich. Ihre zulässige Tiefe ist implementationsbedingt. Auf den aktuellen Wert der Laufkonstanten kann durch die Standardkonstante I zugegriffen werden.

Zusammenfassung:

Das folgende Beispiel, das nahezu alle vorgestellten Sprachelemente benutzt, soll als Zusammenfassung der Einführung in die Syntax der Sprache RPNL dienen. Vertiefend sollen anhand dieses Beispieles zwei Merkmale von RPNL herausgestellt werden:

- die Unterstützung bei der Strukturierung von Programmen und
- die Datenmanipulation von Arrays.

Skatendabrechnung

Zur Aufgabenstellung sei nur kurz erwähnt, daß es sich um ein übliches Verfahren handelt, das Skat zu einem Vierpersonennullsummenspiel macht. Der Algorithmus ist der folgende:

Gewinn/Verlust von (A) := (A-B)+(A-C)+(A-D),

wobei A...D die zu den Spielern (A)...(D) gehörigen Punktezahlen darstellen. Durch zyklische Vertauschung erhält man die Quoten für (B)...(D).

Liest man das eigentliche Hauptprogramm zuerst, das ist hier SKAT, so erkennt man den Aufbau des ganzen Programmes. Es besteht aus folgenden Teilen (Teilprogrammen), die aufgrund ihrer Namen die zugewiesene Aufgabe selbst erklären:

EINGABE
RECHNUNG
SORTIERE
AUSGABE.

Die zur Berechnung und Dokumentation bei der Aufgabe notwendigen Daten werden jeweils in Arrays abgelegt. Diese werden in dem Teilprogramm EINGABE deklariert. Es sind dies einmal die Namen der Mitspieler aber auch deren Punkte. Zur Speicherung der Namen folgendes. In einem Element eines Array können maximal 16 Bit abgelegt werden; das reicht natürlich nicht für einen Namen oder allgemein für Text. Aber es reicht für eine Adresse. Also: in dem Array (hier NAMENSLISTE) werden die Adressen der Texte abgelegt. Dies geht so:

Man vereinbart eine weitere Variable (sie ist nicht unbedingt erforderlich, veranschaulicht aber den Sachverhalt), hier ist es NAME, über die man die Namen einliest

NAME READ(S)

Damit wird der Text (also der Name) unter den Speicherstellen abgelegt, auf die der NAME zeigt. Mit anderen Worten kann man mit der Abfrage NAME ? diese Adresse verfügbar machen und einem Element eines Array zuweisen:

NAME ? I 2 * NAMENSLISTE + :=

Beim Einlesen des nächsten Namens wird NAME aktualisiert und zeigt auf dessen Speicherstellen. Das Teilprogramm RECHNUNG deklariert ein weiteres Array (GELDLISTE), in dem die einzelnen errechneten Beträge vermerkt werden. Zur besseren Handhabung wird der vorne beschriebene Rechnungsmodus in folgende Form gebracht:

GELDLISTE := 3A - B - C - D

Die zyklische Vertauschung, die zur Berechnung der Beträge der anderen drei Spieler notwendig ist, wird durch einen Zähler (XJ) realisiert, der durch eine modulo-Division begrenzt ist.

Nachdem die zustehenden Beträge errechnet sind, werden im Teilprogramm SORTIERE der 1. bis 4. Sieger ermittelt. Dazu werden die Namen der Spieler mit den zugehörigen Daten geordnet; der zugrundeliegende Schlüssel ist der Punktestand. Also, wer die meisten

Punkte hat, der kommt an die Spitze der Liste. Der verwendete Sortieralgorithmus ist äußerst einfach und bedarf keiner Erklärung. Dagegen soll an dieser Stelle auf die Ausführung des TAUSCHvorganges aufmerksam gemacht werden. Anschaulicher können Listenplätze kaum ausgetauscht werden. Die Teilaufgabe AUSGABE ist leicht zu überschauen. Es sei lediglich erwähnt, daß der Mangel an negativen Zahlen künstlich ausgeglichen wird, indem alle Zahlen größer 32768 als negative erklärt werden.

Nachdem die Arrays in dem vorangegangenen Teilprogramm sortiert worden sind, können sie hier einfach der Reihe nach ausgedruckt werden. Dadurch wird der erste Name zum ersten Sieger, der zweite zum zweiten u.s.w.

```
DECLARE
        VAR NAME          VAR HILF          VAR PUNKTE
        VAR XJ            VAR LISTE
        ARRAY NAMENSLISTE 4
        ARRAY PUNKTELISTE 4
        ARRAY GELDLISTE   4
DEND

PROGRAM SPACE
        WRITE '
END

PROGRAM   STERN
        SPACE
        45   1   FOR
                WRITE '+'
        LOOP
END

PROGRAM   2I
        I 2 *
END

PROGRAM   2I+
        I   INC   2   *
END

PROGRAM   EINGABE
        3   0   FOR
                SPACE WRITE 'Wie heißt der'
                I INC HILF :=     HILF PRINT
                WRITE '. Spieler        '
                NAME READ(S)   CR
                NAME ?  2I  NAMENSLISTE  +  :=
                SPACE WRITE 'Wieviele Punkte hat er ?        '
                PUNKTE READ
                PUNKTE  ?  2I  PUNKTELISTE  +  :=
                LOOP
END
```

```
PROGRAM   RECHNUNG
        0  XJ  :=
        REPEAT
                XJ  ?  2  *  PUNKTELISTE  +  ?
                3  *  HILF  :=
                3  1  FOR
                        HILF  ?  I  XJ  ?  +  4  MOD
                        2  *  PUNKTELISTE  +  ?  -  HILF  :=
                      LOOP
                HILF  ?
                XJ  ?  2  *  GELDLISTE  +  :=
                XJ  ?  INC  XJ  :=
        XJ  ?  4  =  UNTIL
        LOOP
END

PROGRAM   TAUSCH
        LISTE  :=
        2I     LISTE  ?  +  ?
        2I+    LISTE  ?  +  ?
        2I     LISTE  ?  +  :=
        2I+    LISTE  ?  +  :=
END

PROGRAM   SORTIERE
        2   0   FOR
                2   0   FOR
                        2I   PUNKTELISTE  +  ?
                        2I+  PUNKTELISTE  +  ?  <  IF
                                PUNKTELISTE  TAUSCH
                                NAMENSLISTE  TAUSCH
                                GELDLISTE    TAUSCH
                                                     THEN
                        LOOP
                 LOOP
END
PROGRAM   AUSGABE
        STERN
        3   0   FOR
                CR
                I   INC   HILF   :=
                SPACE     HILF   PRINT
                WRITE 'Sieger ist   '
                2I   NAMENSLISTE   +  PRINT(S)  CR
                SPACE WRITE 'Er '
                2I   GELDLISTE  +   ?   HILF   :=
                HILF  ?  32768  >  IF
                        0   HILF   ?   -   HILF   :=
                        WRITE 'bezahlt '
                                   ELSE
                        WRITE 'bekommt '
                                   THEN
                HILF PRINT   WRITE ' Taler'
                CR
                LOOP
```

```
        STERN
END

PROGRAM   SKAT
        EINGABE
        RECHNUNG
        SORTIERE
        AUSGABE
END

Ok!
#SKAT
                    Wie heißt der     1. Spieler        Berto

                    Wieviele Punkte hat er ?        486
                    Wie heißt der     2. Spieler        Charly

                    Wieviele Punkte hat er ?        1296
                    Wie heißt der     3. Spieler        Alfred

                    Wieviele Punkte hat er ?        123
                    Wie heißt der     4. Spieler        Dominic

                    Wieviele Punkte hat er ?        2097
                    ++++++++++++++++++++++++++++++++++++++++++++
                         1.Sieger ist   Dominic
                    Er bekommt   4386 Taler

                         2.Sieger ist   Charly
                    Er bekommt   1182 Taler

                         3.Sieger ist   Berto
                    Er bezahlt   2058 Taler

                         4.Sieger ist   Alfred
                    Er bezahlt   3510 Taler
                    ++++++++++++++++++++++++++++++++++++++++++++
```

3.2.4 Die Diskettenbefehle

Erst durch Einführung der Möglichkeit, Daten dauerhaft auf Diskette zu speichern, wird die Aufgabe durchführbar, auch große Datenmengen in sinnvoller Weise zu verarbeiten. Bei der Definition der hierfür notwendigen Prozeduren wurde im wesentlichen auf zwei Kriterien geachtet. Einmal ist dies die Beschränkung auf wenige und in der Anwendung einfache, aber leistungsstarke Befehle. Auf der andere Seite wurde im Hinblick auf die Verarbeitungsgeschwindigkeit ein Möglichstes getan. RPNL arbeitet mit zwei Dateitypen, die sich äußerlich durch den Zusatz im Dateinamen unterscheiden. Dies sind:

- Programmdateien, die Programmtexte enthalten und ausschließlich mit der Prozedur COMPILE verarbeitet werden können. Sie enthalten den Namenszusatz .PRG. Dieser Datei-

typ muß, wenn er in Zusammenhang mit dem Schlüsselwort COMPILE genannt wird, nicht eigens deklariert werden.

- Daten-und Textdateien. Sie tragen den Zusatz .RPN , was eine "verkürzte Abkürzung" für RPNL darstellen soll. Alle in RPNL-Programmen verwendeten Dateien gehören zu diesem Typ. Entsprechend müssen sie auch im Programm deklariert werden.

Die Namenszusätze werden während der Benutzung des RPNL-Compilers nicht benötigt, d.h. sie entfallen bei der Nennung eines Namens. Dennoch sind die oben angeführten Konventionen einzuhalten. Im Zusammenhang mit der Namensgebung der Dateien ist noch folgendes zu beachten. Der erste Buchstabe eines Namens wird als Definition des Laufwerkes, auf dem die zugehörige Datei steht oder stehen soll, aufgefaßt. Dies bedeutet, daß für ein System mit zwei Laufwerken i.a. nur Namen in Frage kommen, die entweder mit A oder B beginnen, nämlich entsprechend der Laufwerksbezeichnung. Darüber hinaus gilt die Einschränkung, die durch das Betriebssystem vorgegeben ist, nämlich die maximale Länge eines Namens mit 8 Zeichen (für CP/M und kompatible Systeme).

Grundsätzlich findet der Datenaustausch bei selbstdefinierten Dateien über jeweils einen zu der Datei gehörigen DMA-Bereich statt. Bekanntlich können nicht einzelne Daten, sondern nur Datenblöcke (Records) übertragen werden. Als definierter Bereich wird dazu der angesprochene DMA-Bereich verwendet. Im allgemeinen ist dies der Adressbereich von 0080h bis 00FFH . Hier bei RPNL verwendet jede Datei ihren eigenen DMA-Bereich. Dessen Basisadresse ist durch Aufruf des Dateinamens erreichbar. Sodann kann dieser Bereich mit Daten gefüllt und mit Hilfe der folgenden Befehle verarbeitet werden.

COMPILE XNAME

Dieser Befehl setzt eine Datei XNAME.PRG auf dem Laufwerk X voraus.

Beispiel:

COMPILE BSORT

sucht eine Datei mit dem Eintrag BSORT.PRG auf dem Laufwerk B. Die zu dem Eintrag XNAME gehörigen Daten werden als Programmtext aufgefaßt. Sie werden von der Diskette eingelesen und gleichzeitig compiliert. Erstellt wurden diese Programme vorher mit einem Texteditor (WORDSTAR, EDDI-80 o.ä.). Genauso sind auch Änderungen an diesen Programmtexten mit einem solchen Editor vorzunehmen. Ein erfolgreicher Übersetzungsversuch wird mit der Meldung

CODE GENERATED

belohnt. Anschließend kann das compilierte Programm mit

seinem Programmnamen (das ist der hinter dem Schlüsselwort PROGRAM) aufgerufen werden. Entdeckt der Compiler einen Fehler, so wird der Übersetzungsvorgang abgebrochen, die Meldung

NO CODE GENERATED

ausgegeben und alle Systemvariablen auf Anfangswerte gesetzt. Im Normalfall wird man anschließend den Compiler mit SYSTEM verlassen, mit dem Editor den Fehler korrigieren und nach dem Starten von RPNL einen erneuten Übersetzungsversuch wagen.

VARF XNAME

Wie alle Namen, die in RPNL als Datenträger fungieren, sind auch Dateinamen dem Compiler durch eine Deklaration mitzuteilen. Dies geschieht durch das Schlüsselwort VARF gefolgt von dem einzuführenden Namen. Bei der Deklaration geschieht folgendes:

- der XNAME wird in die Namensliste eingetragen,
- es wird ein eigener DMA-Bereich mit 80H Länge angelegt,
- mit dem XNAME wird ein FCB eingerichtet.

Wird nach der Deklaration der XNAME in einem Programm genannt, so wird ein Zeiger auf den Anfang bzw das erste Byte des DMA-Bereiches erzeugt. Wird die Datei zur Speicherung von Zahlen benutzt, so wird der DMA-Bereich in geeigneter Weise als Array mit der Länge 80H genutzt.

XNAME OPEN

Diese Prozedur schafft die notwendigen Voraussetzungen für jegliche Arbeit mit Dateien, die auf Diskette gespeichert sind. So ist vor jedem ersten Schreib- oder Lesezugriff für die Datei XNAME die Prozedur OPEN aufzurufen. Damit wird dem System Gelegenheit gegeben, notwendige Eintragungen in dem FCB vorzunehmen.

XNAME CLOSE

Diese Prozedur, die direkt mit OPEN korrespondiert, ist nur nach Schreibzugriffen notwendig. Alle Daten, die in die Datei XNAME geschrieben wurden, sind erst nach dem Aufruf dieser Prozedur dauerhaft gespeichert. Also ist für alle betroffenen Dateien einzelnen nach dem letzten Schreibzugriff die Durchführung dieser Prozedur obligatorisch. Nach Lesezugriffen muß CLOSE nicht aufgerufen werden.

XNAME READF param

Diese Prozedur liest ein Record der benannten Datei XNAME und schreibt diese 80H Byte in den zugehörigen DMA-Bereich. Anschließend ist auf dem Stack ein boolscher Parameter (param) verfügbar, der angibt, ob die letzte Leseoperation erfolgreich war. Oder ob versucht wurde, eine unbeschriebene Datei zu lesen. Letzteres trifft im allgemeinen dann zu, wenn die Datei zu Ende war. In diesem Falle sind auch die Daten im DMA-Bereich nicht gültig. Also bei einem TRUE auf dem Stack war das Lesen erfolgreich und bei einem FALSE eben nicht. Damit ist die Möglichkeit gegeben, Dateien sequentiell in einer REPEAT-Schleife einzulesen, wobei der durch die Prozedur READF übergebene Parameter das Abbruchkriterium darstellt.

XNAME PRINTF

Wurde eine Textdatei mit der Prozedur READF eingelesen und damit die zugehörigen Daten in dem DMA-Bereich abgelegt, so können mit der Prozedur PRINTF diese Zeichen als ASCII-Zeichen interpretiert auf dem Ausgabemedium (Bildschirm und/oder Drucker) ausgegeben werden. Dabei werden immer der gesamte DMA-Bereich, also 80H Zeichen als eine Einheit betrachtet und geschlossen dargestellt. FÜr Dateien, in denen Zahlen gespeichert wurden, ist diese Prozedur nicht geeignet. FÜr diesen Fall muß eine Benutzerroutine geschrieben werden, die den DMA-Bereich als ein Array verwendet und der benutzten Datenstruktur entsprechend die Werte einzeln ausgibt.

NAME XNAME TREADF param1 param2

Wenn mit der Prozedur READF eine Textdatei eingelesen wurde, so stehen davon jeweils 80H Zeichen in dem zugehörigen DMA -Bereich. Damit ist der Text zwar eingelesen, aber für den Compiler bzw den Benutzer noch nicht verfügbar. Deshalb wird durch Aufruf der Prozedur TREADF jeweils ein Datensatz vom DMA-Bereich in den Text-bzw. Variablenbereich des Compilers gebracht. Dabei ist ein Datensatz genau die Folge von Zeichen, die bei dem Zeichen beginnt, das gerade nicht zum vorher übertragenen Satz gehörte, und mit der Folge 0DH 0AH endet. Genau mit dieser Folge wird auch die Prozedur verlassen. Dann sind vier Aktionen durchgeführt:

(1) der aktuelle Datensatz wurde in den Variablenbereich des Compilers übertragen,

(2) die Adresse dieses Satzes im Variablenbereich wurde der Variablen NAME zugewiesen und damit für den Compiler verfügbar gemacht (die Variable NAME muß natürlich vorher deklariert worden sein),

(3) ein Zeiger im DMA-Bereich von XNAME wurde aktualisiert und zeigt auf den nächsten Datensatz,

(4) zwei Parameter wurden auf den Stack gelegt, die u.a. anzeigen, ob das Ende der Datei XNAME erreicht ist; dabei bedeuten:

param1	param2	Erklärung
FALSE	FALSE	Die Datei XNAME ist zu Ende, das Zeichen 1AH wurde erkannt.
FALSE	TRUE	Der aktuelle DMA-Bereich wurde vollständige gelesen und endete mit einem Satzende.
TRUE	TRUE	Der aktuelle DMA-Bereich wurde noch nicht vollständig gelesen, aber das Ende eines Datensatzes ist erreicht.
TRUE	FALSE	Der aktuelle DMA-Bereich wurde vollständig gelesen, aber fällt nicht mit dem Ende eines Datensatzes zusammen. Dies bedeutet, daß nach Erkennen des nächsten Satzendes im neuen DMA-Bereich die Teilstrings aus dem letzten und dem aktuellen Teilstring verschmolzen werden müssen.

XNAME WRITEF

Mit dieser Prozedur wird der gesamte zu XNAME gehörige DMA-Bereich auf Diskette geschrieben. Wurden in diesen Bereich Zahlen oder allgemein Daten, aber kein mit TWRITEF deklarierter Text geschrieben, so ist dieser Aufruf immer gerade dann notwendig, wenn die eingetragenen Daten gespeichert werden sollen. Mit anderen Worten muß das Benutzerprogramm dafür sorgen, daß zum richtigen Zeitpunkt die Prozedur WRITEF gerufen wird. Im Gegensatz dazu wird beim Abspeichern von Text die Prozedur im Bedarfsfalle automatisch gerufen. Aber nur für die Datei, die vorher mit TWRITEF initialisiert wurde. In diesem Falle wird immer dann, wenn genau 80H Zeichen in den zugehörigen DMA-Bereich eingetragen wurden, die Prozedur WRITEF aufgerufen. Dies bedeutet aber auch, daß zum Abschluß jedes Schreibens einmal WRITEF mehr aufzurufen ist, da ja nicht sichergestellt sein kann, daß vorher genau mit einem vollen DMA-Bereich abgeschlossen wurde.

XNAME TWRITEF

Diese Prozedur hat gewissermaßen eine Schalterfunktion. Denn nach deren Aufruf werden alle Zeichen, die auf dem Bildschirm (und/oder Drucker) erscheinen, also von der Tastatur, der Diskette oder andere Programm-Outputs, fortlaufend in den zu XNAME gehörenden DMA-Bereich geschrieben und gespeichert. Damit können auf einfache Art und Weise Textdateien erzeugt werden. Es kann gleichzeitig immer nur eine Datei mit dieser Funktion eingeschaltet werden.

TEND

Diese Prozedur schaltet die mit TWRITEF initiierte Funktion wieder aus. Das bedeutet, daß alle Zeichen, die jetzt auf dem Bildschirm (und/oder Drucker) erscheinen, in keinen DMA-Bereich mehr kopiert werden.

XNAME DELETE

Die Datei mit dem Namen XNAME.RPN wird aus dem Directory der Diskette im Laufwerk X gestrichen und ist somit nicht mehr verfügbar, d.h. auch alle zugehörigen Informationen sind nicht mehr erreichbar. Durch einen anschließenden Aufruf der Prozedur OPEN mit dem gleichen Dateinamen erfolgt wieder ein Eintrag auf der Diskette und eine leere Datei ist neu geschaffen.

SCROFF

Diese Prozedur schaltet den Bildschirm aus. Allerdings nur insofern, als zwar alle Zeichen weiterhin über die entsprechende Ausgaberoutine laufen, also die Prozedur PRINTF in Verbindung mit TWRITEF weiterhin in Funktion bleiben, aber die Anzeige unterdrückt wird.

SCRON

Diese Prozedur hebt SCROFF wieder auf und schaltet den Bildschirm wieder ein.

NOFILES

Diese Aktion schaltet einige der Diskettenbefehle aus. Dadurch werden ablauffähige Programme, die durch die Aktion SAVE NAME erzeugt wurden, kürzer.

Folgende Befehle können nicht mehr ausgeführt werden:

FILES CHGDRIVE DATEINAME READF OPEN CLOSE DELETE
WRITEF PRINTF TWRITEF TREADF TEND ACTDRV
NOFILES

ACTDRV

Diese Variable enthält gemäß CP/M Standard die Nummer des aktuellen Laufwerkes. Da die Variable nur 1 Byte groß ist, kann sie nur mit den Befehlen :=(B) gesetzt bzw. ?(B) abgefragt werden.

CHGDRIVE

Wenn Sie eine Datei bearbeiten wollen, die sich auf einer Diskette in einem anderen als dem aktuellen Laufwerk befindet, so geben Sie diesen Befehl. Es ensteht folgender Dialog:

Aktuelles Laufwerk: A
Gewünschtes Laufwerk: X

Wobei natürlich X Ihrer Eingabe entspricht.

NAME DATEINAME

Wenn NAME ein mit VARF deklarierter Dateiname ist, so haben Sie mit diesem Aufruf die Möglichkeit, in den zugehörigen FCB einen den CP/M Konventionen entsprechenden Namen einzutragen. Dabei muß nach dem Punkt ein Extension mit drei druckbaren Zeichen folgen. Mit der Eingabe des dritten Zeichens nach dem Punkt wird die Eingabe als beendet betrachtet. Mit BACKSPACE können Änderungen an dem Namen angebracht werden. Durch Eingabe von ESCAPE wird der gesamte Name im FCB wieder gelöscht.

3.2.5 Anwendung der Diskettenbefehle

In diesem Abschnitt soll an einigen kurzen Beispielen die Anwendung der Diskettenbefehle gezeigt werden. Damit können mögliche Fragen, die nach dem Studium des Kapitels 3 noch offen geblieben sind, geklärt werden.

3.2.5.1 Erstes Beispiel (EINLESEN)

In diesem Programm wird eine bereits erstellte Textdatei ASORT.RPN, die sich auf dem Laufwerk A befinden muß, eingelesen und gleichzeitig auf dem Bildschirm wieder ausgegeben. Dabei wird eine Wiederholungsschleife benutzt, die solange Datenblöcke einliest, wie READF als Parameter TRUE übergibt. Es soll noch einmal darauf hingewiesen werden, daß die Reihenfolge der folgenden

Prozeduren obligatorisch ist:

VARF	ASORT	-	Deklaration
ASORT	OPEN	-	Öffnen der Datei
ASORT	READF	-	Lesen der Datei
ASORT	PRINTF	-	Ausgeben der Datei

Ein Aufruf der Prozedur CLOSE ist nicht notwendig, da kein Schreibzugriff auf die Datei erfolgte.

```
DECLARE
        VARF ASORT
DEND

PROGRAM EINLESEN
        ASORT OPEN
        REPEAT
                ASORT READF
                NOT UNTIL
                ASORT PRINTF
        LOOP
END
```

3.2.5.2 Zweites Beispiel (KOPIERE)

Das zweite Beispiel geht insofern über das erste hinaus, als während der Ausgabe der Datei alle Zeichen, die auf dem Bildschirm erscheinen, sofort wieder in eine Datei ASORTNEW geschrieben werden. Dies ist dadurch möglich, weil ASORTNEW durch TWRITEF als Textfile initialisiert wurde. Es wurde bereits darauf hingewiesen, daß immer dann, wenn gerade 128 Zeichen auf den Bildschirm geschrieben wurden, diese auf Diskette gerettet werden. Da aber im allgemeinen Fall nicht dann endet, wenn eben gerade 128 Zeichen ausgegeben wurden, ist dafür ein abschließendes WRITEF notwendig.

```
DECLARE
        VARF ASORT      VARF ASORTNEW
DEND

PROGRAM KOPIERE
        ASORT OPEN      ASORTNEW OPEN
        ASORTNEW TWRITEF
        REPEAT
        ASORT READF     NOT UNTIL
                ASORT PRINTF
        LOOP
        TEND
        ASORTNEW WRITEF
        ASORTNEW CLOSE
END
```

```
PROGRAM LIES
        ASORTNEW OPEN
        REPEAT
        ASORTNEW READF   NOT UNTIL
                ASORTNEW PRINTF
        LOOP
END
```

3.2.5.3 Drittes Beispiel (DATEN)

Wurden in den ersten Beispielen die Dateien zur Speicherung von Text benutzt, so werden hier nicht druckbare Zeichen, nämlich Zahlen, gespeichert und anschließend wieder ausgegeben. Dazu wird der DMA-Bereich als Array benutzt, in den die Daten geschrieben werden. Nach Beendigung der Eingabe werden sie auf die Diskette gespeichert.

```
DECLARE
        VARF ADATEN1     VARF ADATEN2
        VAR ZAHL         VAR ZAEHLER
DEND

PROGRAM EINGABE
        ADATEN1 OPEN
        WRITE 'WIEVIELE ZAHLEN ?  '      ZAEHLER READ
        ZAEHLER ? 1 FOR
                WRITE 'ZAHL ?   '        ZAHL READ
                ZAHL   ?   ADATEN1   I    2 *  +  :=
        LOOP
        ZAEHLER   ?   ADATEN1   :=
        ADATEN1 WRITEF
        ADATEN1 CLOSE
END

PROGRAM LESEN1
        ADATEN1 OPEN     ADATEN2 OPEN
        ADATEN1   ?   ZAEHLER  :=
        ZAEHLER   ?   1   FOR
                ADATEN1   I  2 *  +  ?   ZAHL   :=
                ZAHL   PRINT
                I   5   MOD   0=   IF
                        CR
                                     THEN
                ZAHL   ?   ADATEN2   I  2 *  +  :=
                           LOOP
        ZAEHLER  ?  ADATEN2  :=
        ADATEN2 WRITEF
        ADATEN2 CLOSE
END

PROGRAM LESEN2
        ADATEN2 OPEN
        REPEAT
        ADATEN2 READF   NOT   UNTIL
```

```
        ADATEN2   ?   ZAEHLER   :=
        ZAEHLER   ?   1   FOR
                ADATEN2   I 2 *   +    ?   ZAHL   :=
                ZAHL   PRINT
                I   5   MOD   0=   IF
                        CR
                                THEN
                        LOOP
                            LOOP
END

PROGRAM DATEN
        WRITE '     Eingeben          EINGABE'   CR
        WRITE '     Retten            LESEN1 '   CR
        WRITE '     Lesen             LESEN2 '   CR
END
```

3.2.5.4 Viertes Beispiel (NAMEN)

Dieses letzte Beispiel soll noch einmal in einem etwas größeren Zusammenhang die Einsatzmöglichkeit von Textdateien zeigen. Das Programm erstellt eine alphabetisch sortierte Liste von eingegebenen Namen. Diese Liste wird nach der letzten Eingabe ausgegeben und gleichzeitig in der Datei ALISTE gespeichert. Zu Beginn der nächsten Eingabe wird sie eingelesen, die Zeiger auf jeden Namen neu erzeugt und in dem Array NAMENSLISTE abgelegt. Wie das funktioniert, wurde bereits in dem Beispiel "Skatendabrechnung" erklärt. Während des Einlesens wird der Bildschirm mit SCROFF ausgeschaltet. Wenn zum Abschluß die sortierte Liste ausgegeben wird, wird zusätzlich zu jedem Namen eine laufende Nummer erzeugt. Damit diese nicht mit in ALISTE abgespeichert wird, ist die Placierung von TWRITEF und TEND sorgfältig vorzunehmen. Im Übrigen ist das verwendete Sortierverfahren eines der schlechtesten überhaupt. Aber man kann ja nicht immer nur QUICKSORT nehmen.

```
DECLARE
        ARRAY NAMENSLISTE 50    VARF ALISTE
        VAR NAME          VAR ANZAHL
        VAR PARAM1        VAR PARAM2
        VAR ALT           VAR VERSCHMELZEN
        VAR HILF
        STRING BLANK      ' '
        STRING ENDE       '0000'
DEND

PROGRAM INDEX
                I 2 * +
END

PROGRAM EINLESEN
        FALSE VERSCHMELZEN :=
        ALISTE OPEN
        TRUE PARAM1 :=   TRUE PARAM2 :=
```

```
        REPEAT
        ALISTE READF NOT
                PARAM1 ? PARAM2 ? OR  NOT
                OR UNTIL
                REPEAT
                        NAME ALISTE TREADF PARAM1 :=   PARAM2 :=
                PARAM1 ?  PARAM2 ?  AND  NOT UNTIL
                        VERSCHMELZEN ?  NOT IF
                                NAME ? NAMENSLISTE
                                        ANZAHL ? 2 * + :=
                                NAME ? ALT :=
                                ANZAHL ? INC ANZAHL :=
                                  ELSE
                                ALT ? NAME ? +(S) NAME :=
                                NAME ? NAMENSLISTE
                                        ANZAHL ? 2 * + :=
                                  THEN
                                        LOOP
                PARAM1 ? PARAM2 ? NOT AND VERSCHMELZEN :=
                        LOOP
END

PROGRAM NAMEN
        BLANK ? NAMENSLISTE  :=
        1 ANZAHL :=
        SCROFF
        EINLESEN
        SCRON
        WRITE '   ***SORTIEREN EINER NAMENSLISTE***'
        CR CR
        WRITE '    Beenden der Eingabe durch 0000'
        CR CR
        REPEAT
                CR      WRITE '         Name:   '
                NAME READ(S)
        NAME ?  ENDE ?  =(S)   UNTIL
                ANZAHL ? DEC 0 FOR
                        NAMENSLISTE INDEX ?
                        NAME ? >=(S) IF
                                NAME ?
                                NAMENSLISTE INDEX ?
                                NAME :=
                                NAMENSLISTE INDEX :=
                                     THEN
                                LOOP
                NAME ? NAMENSLISTE ANZAHL ? 2 * + :=
                ANZAHL ? INC ANZAHL :=
                                LOOP
        CR
        WRITE '   Geordnete Liste ' CR CR
        ALISTE DELETE   ALISTE OPEN
        ANZAHL ? DEC 1 FOR
                I HILF :=  HILF PRINT  WRITE '   '
                ALISTE TWRITEF
                NAMENSLISTE INDEX PRINT(S) CR
```

```
                TEND
                          LOOP
        ALISTE WRITEF   ALISTE CLOSE
END
```

3.3 Funktionen und Prozeduren in RPNL

Die gesamte Sprache RPNL ist aufgebaut aus Unterprogrammaufrufen oder Funktions-oder Prozedurnamen, so daß es kein eigenes Schlüsselwort (wie CALL oder GOSUB) für diesen Zweck gibt. Schon im Abschnitt 3.1.3 wurde daraufhingewiesen, daß WRITE oder PRINT, aber auch INC oder beispielsweise + (für die Addition) Standardnamen sind. Mit anderen Worten sind dies Prozeduren und Funktionen, also Unterprogramme, die, weil sie häufig und immer wieder benötigt werden, schon vorformuliert und mit einem Standardnamen belegt sind.

Zunächst aber soll in Stichworten der Unterschied zwischen Funktionen und Prozeduren erklärt werden. Also, Funktionen sind Vorschriften, die einem oder mehreren Argumenten höchstens einen Wert zuordnen. So ordnet die Quadratfunktion einer reellen Zahl X ihr Quadrat, nämlich X**2 zu. Formal sieht das in Infix-Notation so aus

```
WERT := X**2.
```

Im Gegensatz dazu sind Prozeduren selbständige Programme und nicht Bestandteile eines Ausdruckes wie Funktionen. Wie aber werden eigene Funktionen und Prozeduren in RPNL formuliert und aufgerufen ? Diese Frage soll hier nicht abschließend behandelt angeschnitten werden. Details werden in dem folgenden Kapitel vermittelt.

Funktionen und Prozeduren sind formal aufgebaut wie Hauptprogramme. Deshalb gilt das Syntaxdiagramm auch in vollem Umfang für Funktionen und Prozeduren. Also sieht ihr Aufbau allgemein so aus:

```
PROGRAM  NAME

     ANWEISUNGSLISTE

END
```

Doch nun zum Aufruf. Wird in dem aufrufenden Programm der Name eines anderen Programms (Funktion oder Prozedur) genannt, so wird zunächst dieses aufgerufene Programm abgearbeitet. Danach erfolgt die Programmfortsetzung mit der Anweisung hinter dem Aufruf.

Insbesondere bedarf die Übergabe der Parametern einiger Erläuterungen. Wie bei einer Sprache, die die UPN benutzt, zu erwarten ist, werden zunächst die aktuellen Parameter genannt und dann der Name der zugehörigen Funktion oder Prozedur. Hierzu ein Beispiel.

Es sei eine Prozedur in PASCAL definiert mit dem Namen AUFRUF, die 2 Parameter (der Datentyp ist hierbei nebensächlich) erwartet. Der Aufruf lautet:

```
AUFRUF(PARA1,PARA2: INTEGER);
```

also ähnlich wie bei anderen Programmiersprachen. Und nun zu RPNL. Hier lautet der entsprechende Aufruf:

```
PARA2 ?  PARA1 ?  AUFRUF
```

An dieser Stelle bietet es sich an, genauer darauf einzugehen, wie letztlich die Parameterübergabe bei den Aufrufen von statten geht. Wir wollen dies an einigen Beispielen demonstrieren. Zunächst werden wir eine Funktion schreiben. Dazu wir greifen das obige Beispiel der Quadratzahlen auf. Im vorliegenden Fall soll das ARGUMENT quadriert, verdoppelt und anschließend einer Variablen WERT zugewiesen werden.

```
DECLARE
  VAR ARGUMENT   VAR WERT
DEND

PROGRAM 2QUADRATFUNKTION
        DUP * 2 *
END

PROGRAM HAUPT1
        ARGUMENT READ CR CR
        ARGUMENT ? 2QUADRATFUNKTION WERT :=
        WERT PRINT
END
```

Charakteristisch ist, daß der errechnete Funktionswert einer Variablen zugewiesen wird. Die Übergabe des Argumentes bzw des Funktionswertes geschieht immer über den Stack. So wird mit

```
ARGUMENT  ?
```

der eingelesene Wert für ARGUMENT auf den Stack gelegt und dann innerhalb der Abarbeitung der Funktion dort weggenommen, entsprechend der Funktionsvorschrift manipuliert und anschließend dort wieder für die Übergabe abgelegt. Genauso läuft die Parameterübergabe bei Prozeduren. Hier ein Beispiel.

```
DECLARE
        VAR ARGUMENT
DEND

PROGRAM STERNPROZEDUR
        0 FOR
                WRITE '*'
        LOOP
END
```

```
PROGRAM HAUPT2
        ARGUMENT READ CR CR
        ARGUMENT ? STERNPROZEDUR
END
```

Wie bereits angedeutet, besteht der Unterschied zur Funktion darin, daß kein Funktionswert berechnet und einer Variablen zugewiesen wird.

Zum Abschluß noch eine weitere Besonderheit. Das Unterprogramm (Funktion oder Prozedur) ist im Gegensatz zu anderen Programmiersprachen nicht Bestandteil des aufrufenden Programms. Es muß dem Syntaxdiagramm entsprechend als eigenständiges (vollständiges) Programm dem aufrufenden Programm vorangestellt werden. Auf unser erstes Beispiel bezogen sähe das so aus:

```
PROGRAM AUFRUF
      .
      .
      .
END

PROGRAM HAUPT
      .
      .
PARA2 ? PARA1 ?  AUFRUF
      .
      .
END
```

3.4 Rekursionen in RPNL

Algorithmen, die die wiederholte Ausführung einer Aktion verlangen, werden meist mithilfe der Wiederholungsanweisungen realisiert. Oftmals liegt aber der Funktion oder Prozedur eine Definition zugrunde, die eine rekursive Lösung nahelegt oder gar verlangt.

Zunächst, was ist rekursiv? Rekursiv bedeutet, daß das Programm auf sich selbst zurückgreift. Mit anderen Worten soll ein Programm wiederum das gleiche Programm aufrufen. Wie das möglich ist, wollen wir uns überlegen. Programme mit einer rekursiven Lösung des Problems haben eine typische Struktur. Sie lösen das Problem durch Lösen mehrerer kleiner Teilprobleme, die sich letztenendes alle auf ein oder wenige Anfangsprobleme zurückführen lassen. Die Ergebnisse zu diesen Anfangsproblemen sind explizit angeggeben. Das Programm setzt dann anschließend alle Teillösungen zu einer Gesamtlösung zusammen. Das führt uns zu der allgemeinen Formulierung von rekursiven Algorithmen:

```
PROGRAM REKURSIV
     ANWEISUNGSLISTE1
     BEDINGUNG IF
          LÖSUNG DES ANFANGSPROBLEMS
               ELSE
          ANWEISUNGSLISTE2
          REKURSIV
          ANWEISUNGSLISTE3
               THEN
END
```

In der Anweisungsliste 2 werden die Argumente für den nächsten Aufruf modifiziert und in der Liste 3 dann die Teillösungen zusammengesetzt. Es ist oft so, daß eine oder zwei dieser Anweisungsfolgen leer sind. Das hängt natürlich individuell von der Struktur des Problems ab. Offensichtlich aber kommt der Bedingung, die zur Lösung des Anfangsproblems führt, eine besondere Bedeutung zu. Denn sie stellt den Rekursionsanfang dar und legt damit die Rekursionstiefe fest. Es muß mit der Anweisungsliste 2 dafür gesorgt werden, daß zum richtigen Zeitpunkt die Bedingung erfüllt wird. Ein einfaches Programmbeispiel soll diesen Sachverhalt verdeutlichen.

```
DECLARE
        VAR ZAHL
DEND

PROGRAM MINUS1
        WRITE 'Zahl : '          ZAHL PRINT      CR
        DEC ZAHL :=
        ZAHL ? 0 <> IF
                ZAHL ? MINUS1
                    THEN
END

PROGRAM HAUPT
        WRITE 'Zahl ? '
        ZAHL READ
        ZAHL ? MINUS1
END
```

Ruft man HAUPT mit dem Argument 5 auf, so erzeugt die Prozedur MINUS1 den folgenden Ausdruck:

```
Zahl :      5
Zahl :      4
Zahl :      3
Zahl :      2
Zahl :      1
```

Man kann daraus erkennen, mit welchen Argumentwerten MINUS1 aufgerufen wird bzw. sich selbst aufruft. Offensichtlich ist das Abbruchkriterium dann erfüllt, wenn ZAHL gleich Null ist. Ein weiteres nicht ganz so triviales Beispiel berechnet Fibonacci-Zahlen. Diese Folge ist definiert:

```
f(1) := 1
f(2) := 1
f(n) := f(n-1)+f(n-2)
```

Offensichtlich entsteht der aktuelle Wert aus der Summe seiner beiden Vorgänger. Nachfolgend ist das zugehörige Programm mit dem Output für das Argument 24 aufgelistet. Wir wollen anschließend überlegen, wie die Verarbeitung also die einzelnen Aufrufe und die Zusammensetzung der Teillösungen realisiert sind.

```
DECLARE
        VAR ARGUMENT     VAR FIBONEU
        VAR FIBOALT      VAR ZAEHLER
        STRING SPACE '           '
DEND

PROGRAM FIBO
        DUP ARGUMENT :=
        1 =  IF
                1 FIBONEU :=
                CR SPACE PRINT(S)
                FIBONEU PRINT   2 ZAEHLER :=
        ELSE
                ARGUMENT ? DEC
                FIBO
                FIBONEU ? FIBOALT ? + FIBONEU :=
                FIBONEU ?       FIBOALT ?
                FIBONEU :=      FIBOALT :=
                FIBONEU PRINT
                ZAEHLER ? 5 MOD 0= IF
                        CR      SPACE PRINT(S)
                THEN
                ZAEHLER ? INC ZAEHLER :=
        THEN
END

PROGRAM FIB
        ARGUMENT READ
        1 FIBOALT :=
        ARGUMENT ? DEC FIBO
        FIBOALT PRINT
END

Ok!

#FIB

24

                1     1     2     3     5
                8    13    21    34    55
               89   144   233   377   610
              987  1597  2584  4181  6765
            10946 17711 28657 46368
Ok!
```

Wie schon angedeutet skizzieren wir nun, welche Aktionen beim Aufruf mit dem Argument 5 in zeitlicher Abfolge ablaufen. Dazu stellen wir die Prozedur FIBO als Block dar mit den jeweils aktuellen Werten der einzelnen Variablen.

```
I-------------------------------------------------------------I
I      ARGUMENT: 4                                            I
I  0   FIBOALT : 1                                            I
I      FIBONEU : /                                            I
I.............................................................I
I                                                             I
I              ----------------------------------------------I
I              I      ARGUMENT: 3                            I
I              I   1  FIBOALT : 1                            I
I              I      FIBONEU : /                            I
I              I.............................................I
I              I                                              I
I              I              ---------------------------------I
I              I              I     ARGUMENT: 2               I
I              I              I  2  FIBOALT : 1               I
I              I              I     FIBONEU : /               I
I              I              I................................I
I              I              I                                I
I              I              I              ------------------I
I              I              I              I     ARGUMENT: 1 I
I              I              I              I  3  FIBOALT : 1 I
I              I              I              I     FIBONEU : 1 I
I              I              I              ------------------I
I              I              I      FIBONEU: 2 ; ADDITION     I
I              I              I      FIBONEU: 1 ; UM--         I
I              I              I      FIBOALT: 2 ; SPEICHERUNG  I
I              I              ---------------------------------I
I              I      FIBONEU: 3 ; ADDITION                   I
I              I      FIBONEU: 2 ; UM-                        I
I              I      FIBOALT: 3 ; SPEICHERUNG                I
I              ----------------------------------------------I
I     FIBONEU: 5 ; ADDITION                                   I
I     FIBONEU: 3 ; UM-                                        I
I     FIBOALT: 5 ; SPEICHERUNG                                I
I                                                             I
I-------------------------------------------------------------I
I                                                             I
I     FIBOALT: 5 ; AUSGABE IM HAUPTPROGRAMM                   I
I                                                             I
I-------------------------------------------------------------I
```

(Die Werte, die ausgegeben werden, sind unterstrichen)

Offensichtlich ist die größte Rekursionstiefe dann erreicht, wenn die Variable ARGUMENT den Wert N11 angenommen hat. Dann nämlich wird FIBONEU mit eins initialisiert. Anschließend werden die Teillösungen genausooft zusammengesetzt wie FIBO aufgerufen wurde. Und dies immer mit den Werten, die zuletzt berechnet wurden. Es soll an dieser Stelle nicht verschwiegen werden, daß sich nicht jede rekursive Definition auch für eine solche Realisierung

eignet. Mit anderen Worten ist die Rekursion nicht immer die optimale Lösung. Andererseits ist sie ein Instrument, das, wenn es richtig eingesetzt wird, effiziente und natürlich strukturierte Programme liefert. Nachfolgend werden zwei weitere Beispiele für rekursive Algorithmen vorgestellt.

3.4.1 Berechnung von Binomialkoeffizienten

Aufgabe: Eine Funktion BINOMIAL berechnet Binomialkoeffizienten n über k. Die Argumente n und k werden als TOP und TOP-1 übergeben, das Ergebnis wird als Variable BIN abgespeichert. Mit einem Programm BINO soll die Funktion BINOMIAL aufgrufen und getestet werden.

Algorithmus: Folgende Definition der Binomialkoeffizienten wird benutzt:

$$\binom{n}{k} \begin{cases} 1 & ; \text{ wenn } k = 0 \text{ oder } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & ; \text{ wenn } 0 < k < n. \end{cases}$$

Eine solche Erklärung der Binomialkoeffizienten legt eine rekursive Lösung des Problems nahe.

Das Programm

Offensichtlich muß das Programm auf folgendes hinauslaufen: BINOMIAL wird sooft aufgerufen, bis das Anfangsproblem erreicht ist. Das ist hier wohl dann der Fall, wenn k=0 oder k=n ist. Ansonsten setzt sich die Lösung (der Funktionswert BINOMIAL) aus den Funktionswerten seiner beiden Vorgänger zusammen. So zeigt Bild 13 die für die Berechnung des Funktionswertes von n=6 und k=4 notwendigen Aufrufe. An jedem Knoten sind die aktuellen Argumentwerte angegeben. Von dort gehen dann zwei oder aber kein neuer Zweig ab. Letzteres da, wo entsprechend der vorn angegebenen Vorschrift der Fall k=0 oder k=n eingetreten ist. Dort ist der Funktionswert gleich 1. Im Programm muß das so realisiert werden, daß immer dann, wenn ein solcher Endpunkt erreicht ist, ein Zähler inkrementiert wird. Nach diesen Vorüberlegungen kann die Funktion BINOMIAL umgangssprachlich formuliert werden:

```
PROGRAMM BINOMIAL
     DEKLARATION der Variablen
     ÜBERNAHME der Argumente (n,k)
     WENN  k=n   oder   k=0,
          DANN erhöhe Zähler um eins,
     SONST
          RUFE BINOMIAL mit den Argumenten (n-1 , k-1) auf,
          RUFE BINOMIAL mit den Argumenten (n-1 , k  ) auf,
     ÜBERGABE der Argumente (Zähler),
ENDE.
```

Bei der Bezeichnung der Variablen halten wir uns eng an die Definition, so daß n und k erhlaten bleiben. Den Zähler wollen

wir BIN nennen. Damit ergibt sich folgende Deklaration:

```
VAR  N     VAR  K     VAR  BIN.
```

Noch ein Wort zum Zähler BIN. Sein Anfangswert muß null sein. Dies ist auch nach der Deklaration der Fall. Nach dem ersten Programmablauf aber nicht mehr. Dann muß er explizit auf null gesetzt werden. Dies kann nicht in Binomial selbst geschehen, da sonst BIN bei jedem Aufruf null gesetzt würde und der Funktionswert immer höchstens 1 sein könnte. Deshalb wird der Anfangswert von BIN im aufrufenden Programm null gesetzt. Der Rest des Programms kann fast wörtlich von unserer umgangssprachlichen Formulierung übernommen werden. Damit ergibt sich:

```
DECLARE
        VAR N  VAR K  VAR BIN
        VAR NN  VAR KK
DEND

PROGRAM BINOMIAL
        N :=  K :=
        N ?  K ?  =
        K ?      0=  OR  IF
                BIN ?  INC  BIN  :=
        ELSE
                K ? DEC  N ?  DEC
                K ?      N ?  DEC
                BINOMIAL
                BINOMIAL
        THEN
END
```

Das aufrufende Programm BINO liest die Argumentwerte ein, übergibt sie BINOMIAL und gibt mit knappem Kommentar das Ergebnis aus.

```
PROGRAM BINO
        0  BIN  :=
        N  READ  N ?  NN  :=
        K  READ  K ?  KK  :=
        K ?  N ?  BINOMIAL
        NN PRINT  WRITE ' UEBER '
        KK PRINT  WRITE ' IST GLEICH '
        BIN PRINT
        CR
END
```

An dieer Stelle sei dem Leser empfohlen, die Bewegungen auf dem Stack für einen Aufruf wie beispielsweise n=4 und k=2 nach dem Schema darzustellen, das auch bei der Quadratwurzelberechnung benutzt wurde. Zur Illustration rekursiver Aufrufe wurde o.a. Programm dahingehend erweitert, daß bei jedem Aufruf von BINOMIAL die aktuellen Argumente ausgedruckt wurden. Hier der Weg von BINOMIAL von der Wurzel bis zum Blatt (in Analogie zu Bild 13).

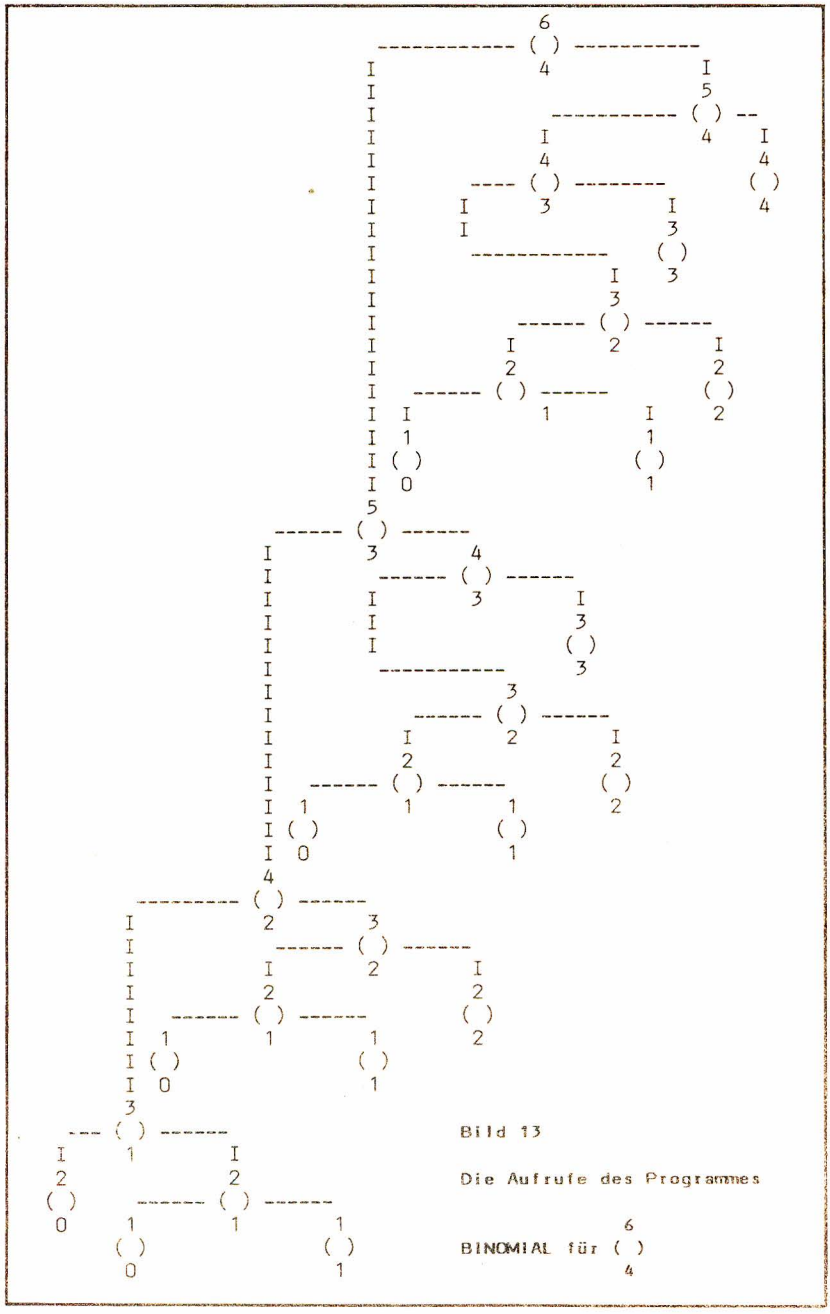


Bild 13

Die Aufrufe des Programmes

BINOMIAL für ( )

```
Ok!
#BINO
4    2
       N=     4       K=     2
       N=     3       K=     2
       N=     2       K=     2       BIN=     1
       N=     2       K=     1
       N=     1       K=     1       BIN=     2
       N=     1       K=     0       BIN=     3
       N=     3       K=     1
       N=     2       K=     1
       N=     1       K=     1       BIN=     4
       N=     1       K=     0       BIN=     5
       N=     2       K=     0       BIN=     6
     4 UEBER      2 IST GLEICH      6

Ok!
#
```

Der Vollständigkeit halber sollen auf noch die Änderungen an dem Programm BINOMIAL gezeigt werden. An dem aufrufenden BINO hat sich nichts geändert.

```
DECLARE
        VAR N  VAR K  VAR BIN
DEND

PROGRAM BINOMIAL
        N :=  K :=
        WRITE '       N='     N PRINT
        WRITE '       K='     K PRINT
        N ?  K ?  =
        K ?      0=  OR  IF
                BIN ?  INC  BIN  :=
                WRITE '       BIN='   BIN PRINT  CR
        ELSE
                CR
                K ? DEC  N ?  DEC
                K ?      N ?  DEC
                BINOMIAL
                BINOMIAL
        THEN
END
```

3.4.2 Der Turm von Hanoi

Es soll noch ein weiteres Problem mithilfe einer Rekursion gelöst werden. Es ist dies die alte indische Sage, daß Mönche seit Beginn der Zeitrechnung damit beschäftigt seien, einen Stapel von 50 goldenen Scheiben mit nach oben hin abnehmendem Durchmesser von einem Pfeiler auf einen zweiten umzuschichten. Dabei dürfen sie einen dritten als Hilfspfeiler benutzen. Sie müssen aber darauf achten, daß niemals eine Scheibe mit größerem Durchmesser auf eine mit kleinerem zu liegen kommt.

Das Programm
Wie bereits angedeutet, wurde dieses Problem durch Rekursion gelöst. Die Prozedur ruft sich mit jeweils "umgeschichteten" Parametern solange auf, bis keine Scheiben mehr da sind. Die jeweils aktuellen Scheiben werden vom Programm Scheibe als Teillösung ausgegeben.

```
DECLARE
        VAR N   VAR XI   VAR XJ   VAR XK
DEND

PROGRAM   SCHEIBE
        WRITE '          Bewege oberste Scheibe vom Pfeiler '
        XI   PRINT     CR
        WRITE '          zum Pfeiler '
        XJ   PRINT     CR CR
END

PROGRAM   UMSCHICHTEN
        N :=   XI :=   XJ :=   XK :=
        N  ?  0  >  IF
                N  ?  XI  ?  XJ  ?  XK  ?
                XJ ?  XK  ?  XI  ?  N   ?   DEC  UMSCHICHTEN
                XK :=  XJ :=  XI :=  N :=
                SCHEIBE
                XI  ?  XJ  ?  XK  ?  N  ?   DEC  UMSCHICHTEN
        THEN
END

PROGRAM   HANOI
        WRITE '          Anzahl der Scheiben ?      '
        N   READ   CR   CR
        3 2 1   N ?   UMSCHICHTEN
END
```

Hier ein Testlauf für vier Scheiben. Es ist nahezu erschreckend, wie die Anzahl der notwendigen Umschichtungen mit der Zahl der Scheiben "explodiert".

```
Ok!
#HANOI
          Anzahl der Scheiben ?     4

          Bewege oberste Scheibe vom Pfeiler      1
          zum Pfeiler      3

          Bewege oberste Scheibe vom Pfeiler      1
          zum Pfeiler      2

          Bewege oberste Scheibe vom Pfeiler      3
          zum Pfeiler      2

          Bewege oberste Scheibe vom Pfeiler      1
          zum Pfeiler      3
```

```
Bewege oberste Scheibe vom Pfeiler      2
zum Pfeiler      1

Bewege oberste Scheibe vom Pfeiler      2
zum Pfeiler      3

Bewege oberste Scheibe vom Pfeiler      1
zum Pfeiler      3

Bewege oberste Scheibe vom Pfeiler      1
zum Pfeiler      2

Bewege oberste Scheibe vom Pfeiler      3
zum Pfeiler      2

Bewege oberste Scheibe vom Pfeiler      3
zum Pfeiler      1

Bewege oberste Scheibe vom Pfeiler      2
zum Pfeiler      1

Bewege oberste Scheibe vom Pfeiler      3
zum Pfeiler      2

Bewege oberste Scheibe vom Pfeiler      1
zum Pfeiler      3

Bewege oberste Scheibe vom Pfeiler      1
zum Pfeiler      2

Bewege oberste Scheibe vom Pfeiler      3
zum Pfeiler      2
```

Ok!

4. Anwendungen der Programmiersprache RPNL

Frei nach dem Motto "Probieren geht über Studieren", sollen, nachdem Syntax und Semantik von RPNL vorgestellt sind, eventuelle offene Fragen geklärt und gleichzeitig Anwendungsmöglichkeiten eröffnet werden.

Dazu werden einige Beispielprogramme entwickelt, die nach steigendem Schwierigkeitsgrad geordnet sind. Sind die ersten Programme noch ausführlich durch begleitenden Text dokumentiert, so wird bei den letzten lediglich noch der Algorithmus bzw die verwendete Datenstruktur kurz erläutert. Dagegen wird vielmehr von dem Vorteil Gebrauch gemacht, daß diese Programme selbstdokumentierend sind. Zwei Dinge ermöglichen dies.

Einmal ist es die Namensgebung der einfachen Zeiger. Dabei kann der Programmierer durch ein wenig Phantasie bei der Wortwahl die Anschaulichkeit (Transparenz der Funktion) der Variablen wesentlich steigern. Auch das Bemühen um eine Reduzierung der Anzahl der Variablen auf ein vernünftiges Maß macht ein Programm übersichtlicher.

Das zweite Hilfsmittel zur Dokumentation ist zunächst vorgegeben durch die Sprachmittel, die eine strukturierte Programmierung unterstützen.Augenscheinlich wird diese Programmierweise durch eine besondere Schreibart, die nämlich Schleifen und Bedingungen als solche sofort erkennen läßt. Damit wird die Tiefe einer Verschachtelung sichtbar und gleichzeitig der Anfang und das Ende von Schleifen. In diesem Zusammenhang soll auch auf die Möglichkeit der Herauslösung von Prozeduren und Funktionen hingewiesen werden. Denn auch mit diesem Mittel können Teilprobleme ausgeklammert werden, die dann im aufrufenden Programm lediglich durch Nennen eines anschaulichen Namens und gleichzeitiger Übergabe der aktuellen Parameter anschaulich gelöst werden.

4.1 Sortieren eines Array

Das Programm dient insbesondere dem Zweck, einige Hinweise zur Handhabung des Array zu geben. Darüberhinaus kann es schließlich ein Beispiel dafür sein, wie eine Aufgabe strukturiert werden kann und in welchem Maße dadurch (hoffentlich) die Lesbarkeit des Programmes fÜr denjenigen steigt, der es nicht selbst erstellt hat.

Aufgabe: Es soll eine Liste von maximal 300 Integerzahlen eingelesen werden. Anschließend wird diese Liste sortiert wieder ausgegeben.

Algorithmus: Beim Sortieren wird der Algorithmus des direkten Einfügens benutzt. Ganz grob läuft folgendes ab. Man betrachtet zunächst nur das erste Element der Liste. Diese Teilliste ist sortiert. Dann wird das nächste (zweite) Element in diese Liste an der entsprechenden Stelle eingefügt. Worauf auch diese Liste sor-

tiert ist. Anschließend wird das nächste Element einsortiert u.s.f.

4.1.1 Das Programm

Die Listenelemente werden in einem Array abgespeichert, das in einem Programm INDEX deklariert wird. Da die maximale Listenlänge 300 betragen soll, wird das Array daraufhin festgelegt. Anhand der Eingabe der einzelnen INTEGER soll gezeigt werden, wie auf ein einzelnes Listen- bzw. Arrayelement zugegriffen wird. In dem entsprechenden Programmteil heißt es:

```
  FOR
I  INDEX  READ
  LOOP   .
```

Offensichtlich ist es so, daß Index auf das Array zeigt und der Laufindex I das daraus entsprechende Element angibt (daß es sich hierbei um einen Laufindex handelt, ist ohne Bedeutung). Um den Zusammenhang vollends aufzudecken, wollen wir die Funktion INDEX ausführlich hinschreiben

```
        FOR
I  2  *  LISTE  +  READ
        LOOP  .
```

Bekanntlich generiert ein einfacher Zeiger wie LISTE eine Adresse. Hier ist es die Basisadresse des Array desgleichen Namens. Basisadresse meint das erste (besser nullte) Element. Dieses Element ist 16 Bit, also zwei Byte lang. Demzufolge erreicht man das nächste Element zwei Adressen weiter. Also

```
2  LISTE  +  READ  ,
```

wenn man es einlesen will. Soll dagegen nicht das nächste, sondern das Ite Element eingelesen werden, so muß offenbar I * 2 Adressen weitergegangen werden. Dadurch kommt man zu

```
I  2  *  LISTE  +  READ  .
```

Zieht man den Teil, den man öfter braucht, heraus und nennt ihn, damit er auch anschaulich bleibt, INDEX, so erhält man wie gehabt

```
I   INDEX   READ   .
```

Und hier das Gesamtprogramm:

```
DECLARE
        ARRAY   LISTE  300
        VAR ANZAHL
        VAR   SJ     VAR   X
DEND
```

```
PROGRAM         INDEX
        2   *   LISTE   +
END

PROGRAM         DRUCKE
        ANZAHL   ?   1   FOR
                 I   INDEX   PRINT
                 I   7   MOD   0=   IF
                         CR
                 THEN
        LOOP
END

PROGRAM         EINGABE
        WRITE 'ANZAHL : '
        ANZAHL   READ
        ANZAHL   ?   1   FOR
                 I   INDEX   READ
        LOOP
END

PROGRAM         SORT
        ANZAHL   ?   2   FOR
                 I   INDEX   ?   X   :=
                 X   ?   0   INDEX   :=
                 I   DEC   SJ   :=
                 REPEAT
                         X  ?   SJ  ? INDEX ? < NOT UNTIL
                         SJ   ?   INDEX   ?
                         SJ   ?   INC   INDEX   :=
                         SJ   ?   DEC   SJ   :=
                 LOOP
                 X   ?   SJ   ?   INC   INDEX   :=
        LOOP
END

PROGRAM    EINAUS
        EINGABE
        DRUCKE
END

PROGRAM    SORTIERE
        EINAUS
        SORT
        CR   DRUCKE
END
```

4.1.2 QUICKSORT

Im folgenden wird ein weiteres Sortierprogramm angegeben, das den gleichen Zweck erfüllt wie das obige. Allerdings ist dieser Algorithmus hinsichtlich der Verarbeitungsgeschwindigkeit überlegen. Auf den Algorithmus als solchen soll hier nicht eingegangen werden. Er ist in der Literatur oft genug erklärt. Vielmehr soll

hier ein Vergleich zwischen beiden Sortiermethoden durchgeführt werden.

Zunächst das Programm QUICKSORT. Es unterscheidet sich an 2 Punkten von dem obigen Programm. Einmal ist dies der Sortierteil selbst (PROGRAM SORT) und dann noch dessen Aufruf.

```
DECLARE
        ARRAY   LISTE   300
        VAR ANZAHL
        VAR   SI      VAR   SJ      VAR   X
        VAR   L       VAR   R
DEND

PROGRAM         INDEX
        2   *   LISTE   +
END

PROGRAM         DRUCKE
        ANZAHL   ?   1   FOR
                 I   INDEX   PRINT
                 I   7   MOD   0=   IF
                         CR
                 THEN
        LOOP
END

PROGRAM         EINGABE
        WRITE 'ANZAHL : '
        ANZAHL   READ
        ANZAHL   ?   1   FOR
                 I   INDEX   READ
        LOOP
END

PROGRAM        SORT
        DUP   SI   :=     L   :=
        DUP   SJ   :=     R   :=
        SI   ?   SJ   ?   +   2   DIV   INDEX   ?   X   :=
        REPEAT
                 REPEAT
                         SI   ?   INDEX   ?   X   ?   <
                 NOT   UNTIL
                         SI   ?   INC   SI   :=
                 LOOP
                 REPEAT
                         X   ?   SJ   ?   INDEX   ?   <
                 NOT UNTIL
                         SJ   ?   DEC   SJ   :=
                 LOOP
                 SI   ?   SJ   ?   <=   IF
                         SI   ?   INDEX   ?
                         SJ   ?   INDEX   ?
                         SI   ?   INDEX   :=
```

```
                    SJ   ?   INDEX   :=
                    SI   ?   INC  SI   :=
                    SJ   ?   DEC  SJ   :=
            THEN
        SI   ?   SJ   ?   >   UNTIL
        LOOP
        SI   ?   R   ?
        L   ?   SJ   ?   <   IF
                SJ   ?   L   ?   SORT
        THEN
        R   :=      SI   :=
        SI   ?   R   ?   <   IF
                R   ?   SI   ?   SORT
        THEN
END

PROGRAM    EINAUS
        EINGABE
        DRUCKE
END

PROGRAM    QUICKSORT
        EINAUS
        ANZAHL   ?   1   SORT
        CR   DRUCKE
END
```

Als Vergleichskriterium soll die Anzahl der notwendigen Vertauschungen dienen, die von dem Sortierprogramm durchgeführt werden müssen. Um für beide Programme die gleichen Startbedingungen zu haben, wird die Eingabe jeweils durch einen Zufallszahlengenerator simuliert, der beide Programme mit den gleichen Zahlen füttert. Offensichtlich ist dann derjenige der Schnellere, der die wenigsten Vertauschungen benötigt. Denn die brauchen natürlich Rechenzeit. Zunächst das Programm.

```
DECLARE
        ARRAY   LISTE   300
        CONST   ANZAHL  300
        VAR   ZUF1      VAR   ZUF2
        VAR   SJ        VAR   X
        VAR ZAEHLER1
        VAR   SI        VAR   L      VAR   R
        VAR ZAEHLER2
DEND

PROGRAM     RANDOM
        ZUF1   ?   31   *   1870   MOD   1   +   ZUF1   :=
        ZUF2   ?   29   *   1729   MOD   1   +   ZUF2   :=
        ZUF2   ?   5   DIV   ZUF1   ?   13   DIV   *
END
```

```
PROGRAM         INDEX
        2   *   LISTE   +
END

PROGRAM         DRUCKE
        CR
        WRITE '        '
        ANZAHL   1   FOR
                I   INDEX   PRINT
                I   9   MOD   0=   IF
                        CR   WRITE '        '
                THEN
        LOOP
END

PROGRAM         EINGABE
        123   ZUF1   :=     234   ZUF2   :=
        ANZAHL   1   FOR
                RANDOM   I   INDEX   :=
        LOOP
END

PROGRAM         SORT1
        ANZAHL ?   2   FOR
                I   INDEX   ?   X   :=
                X   ?   0   INDEX   :=
                I   DEC   SJ   :=
                REPEAT
                X   ?   SJ   ?   INDEX ?   <   NOT   UNTIL
                        SJ   ?   INDEX   ?
                        SJ   ?   INC   INDEX   :=
                        SJ   ?   DEC   SJ   :=
                        ZAEHLER1   ?   INC   ZAEHLER1   :=
                LOOP
                X   ?   SJ   ?   INC   INDEX   :=
        LOOP
END

PROGRAM    SORT2
        DUP   SI   :=     L   :=
        DUP   SJ   :=     R   :=
        SI   ?   SJ   ?   +   2   DIV   INDEX   X   :=
        REPEAT
                REPEAT
                SI   ?   INDEX   ?   X   ?   <
                NOT   UNTIL
                        SI   ?   INC   SI   :=
                LOOP
                REPEAT
                X   ?   SJ   ?   INDEX   ?   <
                NOT UNTIL
                        SJ   ?   DEC   SJ   :=
                LOOP
                SI   ?   SJ   ?   <=   IF
                        SI   ?   INDEX   ?
```

```
                    SJ   ?   INDEX   ?
                    SI   ?   INDEX   :=
                    SJ   ?   INDEX   :=
                    SI   ?   INC   SI   :=
                    SJ   ?   DEC   SJ   :=
                    ZAEHLER2   ?   INC   ZAEHLER2   :=
            THEN
        SI   ?   SJ   ?   >   UNTIL
        LOOP
        SI   ?   R   ?
        L   ?   SJ   ?  <   IF
                SJ   ?   L   ?   SORT2
        THEN
        R   :=      SI   :=
        SI   ?   R   ?   <   IF
                R   ?   SI   ?   SORT2
        THEN
END

PROGRAM    EINAUS
        EINGABE
        DRUCKE
END

PROGRAM    SORTIERE
        0 ZAEHLER1 :=     0 ZAEHLER2 :=
        WRITE 'UNSORTIERTE LISTE'
        EINAUS
        SORT1
        EINGABE
        ANZAHL ?   1   SORT2
        CR
        WRITE 'SORTIERTE LISTE'
        DRUCKE
        CR   CR   CR
        WRITE '   ***   AUSWERTUNG   ***'   CR
        WRITE 'DER ALGORITHMUS DES DIREKTEN
        WRITE 'EINFUEGENS BENOETIGTE'
        ZAEHLER1   PRINT   WRITE ' SCHRITTE' CR
        WRITE 'DER ALGORITHMUS QUICKSORT BENOETIGTE '
        ZAEHLER2   PRINT   WRITE ' SCHRITTE'
END
```

Und hier das Ergebnis des Vergleiches. Man kann daran auch die Qualität des Zufallszahlengenerators erkennen. Der mögliche Einwand, daß die Zufallszahlen zweimal erzeugt und damit unterschiedliche Voraussetzungen für die Sortierprogramme gegeben seien, trifft nicht zu, da der Generator bei gleicher Initialisierung auch die gleiche Folge von Zahlen erzeugt. Zu dem eigentlichen Ergebnis ist kaum etwas hinzuzufügen. QUICKSORT schlägt das andere Verfahren haushoch.

```
Ok!
>SORTIERE
UNSORTIERTE LISTE
        1600   9248    558  17934   6810   1118   2449   2190   6336
        2788  20125  31356  28728   4368   2574  31191    585  13775
        5986   4662  25344   3901  46150   9546  12740   8268  26103
        7420  17745   5644  38936  14904  13975    918  38808  16836
       22230    282  21534  19720   4368   5040   7328   9159  37950
       28441   3146   2295   9447  18249  28260   8712   2256   2680
       33210  22560   2332  21228  18939  10506  23871  19773   3320
       34814   5875   6440  14442  43344   8308  29337  11826   8736
        2668   6422   1859  46648  19320   2232   9280   2412    910
        2912   1344  17263   7950   7654  13114   3300  11760  13260
       14030  12194  32053  10752    638  17871    537   1995  22960
       21090  42636   2988   2556  16168  25970   2158  38872  15529
       14365   3230  42160  21060   8320   7020  36696   8280   4290
         576   5624  18360   6552   6864    576  23478  11155   5474
       38720   7803  13113  22638  20430    858   1488  17082  16236
        3854   9275  20358  23112   1326   8541  32877   7800  14935
        3431   1470  16704   6063  40300  14097  10530  28938  26796
        2660   3549  45152  21980   4968  24940    306   1470   7808
        1026   8601   8730  12470   6162   2100  10992   4828  37950
       32026   3458   9520   1474  26307    942  12540   3854  38073
       35640  19920   6248  20984  17346   8034  36951   8957   5395
        5746  15500  19872  10790   9072   8844  15939  18980    192
        1702  45968   8008   8232   8832   7998   9200  31892  22022
        2457  31584  21791  23850   5874   7968  25740  30135  18330
        6466   7917  25787   3264   2117  18837   7160   2163  13160
        6650  28101   4644   2232  23876  21465   7553  39904   5567
        2873  25840  23800   7020  14848   2340   1390   3840    198
       17568   2280  11610   9243   2860    864  12376  11155   6164
       42560  32368   2046  32634    681   1235   2542  24309  17424
        3403  24850  20124  21168   1014  13221  14893  12675   2465
        9052   4536  12480   1269  42900   7659  16900    636  17094
       19040  15288   7968  10048  17802  24725   4046  35574   6588
       24111  10857  26190   9570   3744  16380  28167   6674  17490
       20793   2782   2890
SORTIERTE LISTE
         192    198    282    306    537    558    576    576    585
         636    638    681    858    864    910    918    942   1014
        1026   1118   1235   1269   1326   1344   1390   1470   1470
        1474   1488   1600   1702   1859   1995   2046   2100   2117
        2158   2163   2190   2232   2232   2256   2765   2295   2332
        2340   2412   2449   2457   2465   2542   2556   2574   2660
        2668   2782   2788   2860   2873   2890   2912   2988   3146
        3230   3264   3300   3320   3403   3431   3458   3549   3744
        3840   3854   3854   3901   4046   4290   4368   4368   4536
        4644   4662   4828   4968   5040   5395   5474   5567   5624
        5644   5746   5874   5875   5986   6063   6162   6164   6248
        6336   6422   6440   6466   6552   6588   6650   6674   6810
        6864   7020   7020   7160   7328   7420   7553   7655   7659
        7800   7803   7808   7917   7950   7968   7968   7998   8008
        8034   8232   8268   8280   8308   8320   8541   8601   8712
        8730   8736   8832   8844   8957   9052   9072   9159   9200
        9243   9248   9275   9280   9447   9520   9546   9570  10048
```

```
10506 10530 10752 10790 10857 10992 11155 11155 11610
11760 11826 12194 12376 12470 12480 12540 12675 12740
13113 13114 13160 13221 13260 13775 13975 14030 14097
14365 14442 14848 14893 14904 14935 15288 15500 15529
15939 16168 16236 16380 16704 16836 16900 17082 17094
17263 17346 17424 17490 17568 17745 17802 17871 17934
18249 18330 18360 18837 18939 18980 19040 19320 19720
19773 19872 19920 20124 20125 20358 20430 20793 20984
21060 21090 21168 21228 21465 21534 21791 21980 22022
22230 22560 22638 22960 23112 23478 23800 23850 23871
23876 24111 24309 24725 24850 24940 25344 25740 25787
25840 25970 26103 26190 26307 26754 26796 28101 28167
28260 28441 28728 28938 29337 30135 31191 31356 31584
31892 32026 32053 32368 32634 32877 33210 34814 35574
35640 36696 36951 37950 37950 38073 38720 38808 38872
38936 39904 40300 42160 42560 42636 42900 43344 45152
45968 46150 46648


   ***   AUSWERTUNG   ***
DER ALGORITHMUS DES DIREKTEN EINFUEGENS BENOETIGTE  22235 SCHRITTE
DER ALGORITHMUS QUICKSORT BENOETIGTE    678 SCHRITTE
Ok!
```

4.2 Die Erzeugung eines 'Magischen Quadrates'

Unter einem 'Magischen Quadrat' versteht man eine n * n - Matrix, deren Komponenten paarweise verschiedene Integerwerte zwischen 1 und n**2 haben. Weiterhin muß die Summe einer jeden Zeile, Spalte oder Diagonalen denselben Wert ergeben. Dazu existiert für ungerades n eine einfache Vorschrift, wie die einzelnen Felder zu besetzen sind:

> Beginne in der Mitte der ersten Zeile mit 1; dann gehe eins noch oben und eins nach links und trage die nächst größere Zahl in das leere Feld ein. Ist das vorliegende Feld schon besetzt, so gehe stattdessen um ein Feld nach unten und trage die Zahl ein. Wird der rechte Rand überschritten, so gehe auf den linken Rand in der gleichen Zeile. Wird der obere Rand überschritten, so gehe auf den unteren Rand in der gleichen Spalte.

Der oben angegebene Algorithmus ist in dem folgenden Programm realisiert. Dabei ist gut die Modularisierung und die Vorteile der strukturierten Programmierung bei der Plausibilitätskontrolle zu erkennen.

```
DECLARE
  VAR GROESSE  VAR VERT  VAR HORI  VAR ZEILE
  STRING TEXT ' MAGISCHES QUADRAT '
  ARRAY QUAD 144
DEND
```

```
PROGRAM SPACE
; Diese Prozedur erzeugt bei Aufruf 20 Blanks
        20  0 FOR
                WRITE ' '
        LOOP
END

PROGRAM  BLANK
; Diese Prozedur erzeugt soviele Blanks, wie der Parameter angibt
; Im folgenden ist unter Parameter die Zahl gemeint,  die vor dem
; Aufruf der Prozedur genannt wird
        80  GROESSE ?  8  *  -  2  DIV  0  FOR
                WRITE ' '
        LOOP
END

PROGRAM  STRICH
; Diese Prozedur erzeugt soviele Striche, wie der Parameter angibt
        0  FOR
                WRITE '-'
        LOOP
END

PROGRAM  STERN
; Diese Prozedur erzeugt soviele Sterne, wie der Parameter angibt
        0  FOR
                WRITE '*'
        LOOP
END

PROGRAM EINGABE
; Diese Prozedur fragt ab, wie groß das Quadrat werden soll
        REPEAT
                CR  SPACE  10  STERN  TEXT PRINT(S)
                10  STERN  CR  CR     SPACE
                WRITE 'WIE GROSS SOLL DAS '
                WRITE 'QUADRAT SEIN (<= 9) ?      '
                GROESSE  READ  CR
; hier findet die Überprüfung auf die gültige Eingabe statt
        GROESSE  ?   9  <=
        GROESSE  ?   2  MOD  1  =  AND  UNTIL
                SPACE
                WRITE 'BITTE UNGERADE ZAHL, DIE KLEINER'  CR
                SPACE
                WRITE 'ALS  9  IST, EINGEBEN:'            CR
        LOOP
END

PROGRAM  ELEMENT
        VERT ?  DEC GROESSE  ? *  HORI  ?  DEC  +  2  *  QUAD  +
END

PROGRAM  G++
        GROESSE ?  DUP  *  DEC
END
```

```
PROGRAM  LOESCHE
; diese Prozedur löscht das Array, in das
; die Zahlen geschrieben werden
        G++  0  FOR
                0  I  2  *  QUAD  +  :=
        LOOP
END

PROGRAM  AUSGABE
; diese Prozedur gibt das Quadrat formatiert aus
        CR
        SPACE
        TEXT  PRINT(S)
        WRITE 'MIT DER LAENGE '  GROESSE  PRINT  CR
        CR    BLANK  GROESSE  ?  8  *  STRICH  CR
        BLANK
        WRITE 'I'
        1  ZEILE  :=
        G++  0  FOR
                I  2  *  QUAD  +  PRINT
                WRITE ' I'
                I  INC  GROESSE  ?  MOD  0=  IF
                        CR  BLANK  GROESSE  ?  8  *  STRICH  CR
                        ZEILE  ?  GROESSE  ?  <  IF
                                BLANK  WRITE 'I'
                                ZEILE  ?  INC  ZEILE  :=
                        THEN
                THEN
        LOOP
END


PROGRAM  QUADRAT
; dies ist das Hauptprogramm, das auch aufgerufen wird
; hier wird das Magische Quadrat erzeugt und
; die Ausgabe veranlaßt
        EINGABE
        LOESCHE
        GROESSE ?  DEC  2  DIV  2  +  HORI  :=
        2  VERT  :=
        G++  INC  1  FOR
                HORI  ?  DEC  HORI  :=
                HORI  ?  0=  IF
                        VERT ? 1 =  IF
                                1  HORI :=
                                3  VERT :=
                        ELSE
                                GROESSE  ?  HORI  :=
                        THEN
                THEN
                VERT  ?  DEC  VERT  :=
                VERT  ?  0=  IF
                        GROESSE  ?  VERT  :=
                THEN
```

```
                ELEMENT  ?  0  <>  IF
                        VERT  ?  2  +  VERT  :=
                        HORI  ?  1  +  HORI  :=
                THEN
                I  ELEMENT  :=
        LOOP
        AUSGABE
END
```

Der Rechner erzeugt dazu den folgenden Ausdruck:

```
Ok!
↘QUADRAT

                ********** MAGISCHES QUADRAT **********

                WIE GROSS SOLL DAS QUADRAT SEIN (<= 9) ?     6

                BITTE UNGERADE ZAHL, DIE KLEINER
                ALS  9  IST, EINGEBEN!

                ********** MAGISCHES QUADRAT **********

                WIE GROSS SOLL DAS QUADRAT SEIN (<= 9) ?     7


                 MAGISCHES QUADRAT MIT DER LAENGE      7

        ---------------------------------------------------------
        I    28 I    19 I    10 I     1 I    48 I    39 I    30 I
        ---------------------------------------------------------
        I    29 I    27 I    18 I     9 I     7 I    47 I    38 I
        ---------------------------------------------------------
        I    37 I    35 I    26 I    17 I     8 I     6 I    46 I
        ---------------------------------------------------------
        I    45 I    36 I    34 I    25 I    16 I    14 I     5 I
        ---------------------------------------------------------
        I     4 I    44 I    42 I    33 I    24 I    15 I    13 I
        ---------------------------------------------------------
        I    12 I     3 I    43 I    41 I    32 I    23 I    21 I
        ---------------------------------------------------------
        I    20 I    11 I     2 I    49 I    40 I    31 I    22 I
        ---------------------------------------------------------

Ok!
↘
```

4.3 Test von Schreiblesespeichern

Bei Zweifel an dem einwandfreien Funktionieren von Schreiblesespeichern ist es zweckmäßig, diese durch Software zu überprüfen. Dabei sind im wesentlichen 3 Dinge zu berücksichtigen:

1. meist sind nur einige Speicherzellen eines Bausteines defekt.
2. Der Fehler tritt nur sporadisch auf und
3. erscheint oftmals nur nach längerer Betriebsdauer.

Diese Kriterien wurden beim Entwurf des Testprogramms folgendermaßen berücksichtigt:

1. Es werden alle Speicherstellen nacheinander geprüft.
2. und
3. Das Programm ist in einer Schleife angelegt, die jeden Baustein 100mal testet.

Man läßt das Programm am besten nach einer gewissen Betriebszeit (ca 1 Stunde) laufen. Wird danach kein Fehler diagnostiziert, so kann man davon ausgehen, daß der getestete Speicherbereich in Ordnung ist. Das vorliegende Programm zeigt einmal mehr, wie nützlich es ist, Programme in Maschinensprache direkt in Hochsprachprogramme einbinden zu können. Es ist naheliegend, daß solche Probleme, die die direkte Adressierung von Speicherbausteinen voraussetzen, am geschicktesten in einer Sprache formuliert werden, die dieser Ebene am nächsten kommt, nämlich die Maschinensprache.

```
DECLARE
  CONST ANZDRCHLAEUFFE  100
  VAR   ERSTE           VAR     LETZTE
  VAR   ZAEHLER         VAR     FEHLER
  VAR   FEHLERSTELLE
DEND

CODE  TEST
   D1                          ;         POP DE
   E1                          ;         POP HL
   97                          ; MARKE1  A    := 0
   77                          ; MARKE2  (HL) := A
   BE                          ;         (HL) = A ?
   01 FF FF                    ;         BC   := FEHLER
                               ;         FF FF = TRUE
   20 15                       ;         JR   NZ,FEHL
   46                          ;         B    := (HL)
   B8                          ;         A    = B ?
   01 FF FF                    ;         BC   := FEHLER
   20 0E                       ;         JR   NZ,FEHL
   3C                          ;         A    := A + 1
   20 EF                       ;         JR   NZ,MARKE2
   23                          ;         HL   := HL + 1
   A7                          ;         CLEAR CARRY
```

```
   E5                          ;        PUSH HL
   ED 52                       ;        SBC  HL,DE
   E1                          ;        POP  HL
   20 E6                       ;        JR   NZ,MARKE1
   01 00 00                    ;        BC   := KEIN FEHLER
                               ;        00 00 = FALSE
   E5                          ; FEHL   PUSH HL :
                               ;            DEFEKTE ZELLE
   C5                          ;        PUSH BC :
                               ;            FEHLER   JA/NEIN
CEND

PROGRAM  RAMTEST
  CR
  WRITE '     Erste Speicherstelle     '
  ERSTE  READ  CR
  WRITE '     Letzte Speicherstelle    '
  LETZTE READ  CR
  0  ZAEHLER  :=
  REPEAT
    ERSTE  ?    LETZTE  ?
    TEST  FEHLER  :=   FEHLERSTELLE  :=
    ZAEHLER  ?  INC  ZAEHLER  :=
  ZAEHLER  ?  ANZDRCHLAEUFFE  =
  FEHLER  ?  TRUE  =              OR  UNTIL
  LOOP
  FEHLER  ?  FALSE  =  IF
    WRITE '     Es wurde zwischen den Speicherstellen '
    ERSTE  PRINT  CR
    WRITE '     und '  LETZTE  PRINT
    WRITE ' kein Fehler gefunden' CR CR
  ELSE
    WRITE '     Es wurde in der Speicherstelle '
    FEHLERSTELLE PRINT
    WRITE ' ein Fehler entdeckt!'
  THEN
END



Ok!
>RAMTEST

     Erste Speicherstelle    40960

     Letzte Speicherstelle   41088

     Es wurde zwischen den Speicherstellen  40960
     und  41088 kein Fehler gefunden


Ok!
```

4.4 Implementierung eines Editors

Nahezu alle Rechnersysteme verfügen über dieses Standardprogramm, das den Benutzer bei der Arbeit mit dem Rechner unterstützen soll. Dabei werden Änderungen und Einfügungen mit Hilfe einfacher aber mächtiger Anweisungen durchgeführt. Die Aufgaben des hier beschriebenen Editorprogrammes sind im wesentlichen:

(1) Einführung eines Editormode in den RPNL-Compiler. Also Eröffnung der Möglichkeit, Programme als Quellprogramme zu speichern und zu redigieren.

(2) Anwendung der Sprache in einem grösseren Zusammenhang.

Der Editor ist zeilenorientiert. Das bedeutet, dass alle Manipulationen anhand von Zeilennummern durchgeführt werden. Allerdings werden sie nicht mit abgespeichert, sondern bei Bedarf neu berechnet. Dies ergibt ein Format, das vom Compiler, der diesen Text ja verstehen muss, akzeptiert wird. Der Text wird in einem sogenannten Editbereich abgelegt. Zu dessen Verwaltung werden einige Zeiger benutzt. Das sind zunächst einmal 2 Zeiger auf die aktuelle Zeile. Dann spielt aber auch der Zeiger auf den Variablenbereich eine wesentliche Rolle. Bei Aufruf des Schlüsselwortes READ(S) werden nämlich die eingegebenen Texte in den Variablenbereich geschrieben, also unter diesem Zeiger abgelegt. Erst später werden sie in den Editbereich kopiert. Der Text wird beendet durch das Zeichen '§'. Dieses Zeichen wird auch mit abgespeichert und bei allen Abbruchkriterien benutzt. Wie schon angedeutet, werden alle Korrekturen mit Zeilennummern eingebracht. Das hat zur Folge, daß häufig solche Nummern als Eingabe verlangt werden. Dies geschieht im Programm mit Hilfe der Prozedur READ , die auf eine numerische Eingabe besteht. Das bedeutet, eine solche Abfrage kann nicht übergangen werden und alle geforderten Angaben sind zu machen, auch wenn sie nicht benötigt werden. Die Beschreibung der einzelnen Aktionen erfolgt in den nächsten Kapiteln.

4.4.1 Das Menue

Die Aufgabe des Hauptprogrammes liegt im wesentlichen darin, die möglichen Aktionen zu definieren und aufzurufen. Doch werden bei jedem erneuten Aufruf des Programmes EDITOR die erwähnten Zeiger auf Anfangswerte gesetzt. Dadurch wird der gesamte Editbereich geleert. Bei Aufruf einer nicht definierten Aktion erfolgt eine Fehlermeldung mit Hinweis auf die erlaubten Prozeduren. Mit der Aktion ENDE wird der Editor verlassen.

4.4.2 Die Prozeduren

4.4.2.1 EINGABE

Diese Aktion ermöglicht es, Texte einmal einzugeben, aber auch

Texte in einem schon bestehenden File einzufügen. Dabei wird immer hinter der eingegebenen Zeilennummer eingefügt. Bei der ersten Eingabe ist die Zeilennummer belanglos, wird aber gefordert. Der Vorgang des Einfügens soll an einem kurzen Beispiel erläutert werden. Der Editbereich habe folgendes Aussehen:

```
3FFE        15 40                 <--- Endadresse
4000 (10)   08 X X X X X X X X
4009 (20)   03 X X X
400D (30)   07 X X X X X X X      <--- Textende
4015        01 40                 <--- Endezeichen
```

(X steht für ein beliebiges ASCII-Zeichen)

Hinter der Zeilennummer 20 soll eingefügt werden. Die Adresse 400D wird als Einfügeadresse ermittelt. In den Variablenbereich wird der einzufügende Text geschrieben, dessen Länge ermittelt und der Teil des Editbereiches von 400D bis 4017 um genau diese Länge nach hinten verschoben. In die nun freie Lücke wird dann der Text aus dem Variablenbereich kopiert.

```
PROGRAMM EINGABE
   LIES Zeilennummer
   STELLE Einfügeadresse fest
   WIEDERHOLE
   BIS Eingabe = '§'
      GIB Zeile ein
      STELLE Länge der Zeile fest
      VERSCHIEBE Editbereich um diese Länge
      KOPIERE Zeile aus dem Variablenbereich nach Einfügeadresse
      EINFUEGEADRESSE ist gleich altem Wert + Länge der Zeile
      KORRIGIERE Variablenadresse und Endadresse
   ENDE
ENDE
```

4.4.2.2 DRUCKE

Nach Eingabe dieser Aktion wird der gesamte Editbereich ausgedruckt. Das Programm läuft in einer Schleife, die bei Erkennen des Terminators '§' abgebrochen wird. Dieses Programm benötigt keine Eingabeparameter.

```
PROGRAMM DRUCKE
   ADRESSE ist der Anfang des Editbereiches
   WIEDERHOLE
   BIS String von Adresse = '§'
      DRUCKE String von Adresse
      NEUE Zeile (CR)
      LADE die Länge dieses String
      ADDIERE sie zu Adresse
   ENDE
ENDE
```

4.4.2.3 LISTE

Im Gegensatz zu DRUCKE werden bei dieser Aktion einmal die Zeilennummer mit ausgegeben und auch nur der Bereich zwischen den eingegebenen Zeilennummern aufgelistet. Damit die Ausgabe langer Programme nicht durchläuft, ohne gelesen werden zu können, werden immer nur 8 Zeilen auf dem Bildschirm dargestellt. Danach erwartet das Programm die Eingabe einer beliebigen Zahl. Ansonsten funktioniert der Algorithmus wie der bei DRUCKE.

```
PROGRAMM LISTE
   ADRESSE ist der Anfang des Editbereiches
   ZEILENNUMMER ist 10
   LIES Anfang (erste Zeilennummer)
   LIES Ende (letzte Zeilennummer)
   WIEDERHOLE
   BIS String von Adresse = '§'
      ODER Zeilennummer > Ende
      WENN Zeilennummer >= Anfang
         DANN drucke Zeilennummer und ' '
            drucke String von Adresse, neue Zeile
            WENN der Quotient aus Zeilennummer und Anfang einen
            Rest von 80 ergibt (entsprechend 8 Zeilen)
               DANN warte auf Eingabe einer Zahl
            ENDE
      ENDE
      LADE die Länge des aktuellen String
      ADDIERE sie zur Adresse
      ERHOEHE die Zeilennummer um 10
   ENDE
ENDE
```

4.4.2.4 LOESCHE

Mit dem Programm LOESCHE können Teile des Editbereiches ausradiert werden. Es ist naheliegend, daß diese Aktion ähnlich funktioniert wie EINGABE, aber natürlich umgekehrt. So werden gem. der eingegebenen Grenzen der zu löschende Speicherbereich durch den verbleibenden überschrieben.

```
PROGRAMM LOESCHE
   ADRESSE ist der Anfang des Editbereiches
   ZEILENNUMMER ist 10
   LIES Anfang (erste Zeilennummer)
   LIES Ende (Letzte Zeilennummer)
   WIEDERHOLE
   BIS der String von Adresse = '§' ist
   ODER die Zeilennummer > Ende
      WENN die Zeilennummer >= Anfang
         DANN
            LADE Quelle (2 mal)
            LADE Ziel und vertausche sie
            LADE Adresse und vertausche
            SUBTRAHIERE (das ist jetzt die Länge)
```

```
            Erhöhe sie um 2 (wegen '§')
            VERSCHIEBE alles
            BERECHNE Adresse
         SONST
            BERECHNE neue Adresse
      ENDE
      ERHOEHE Zeilennummer um 10
   ENDE
ENDE
```

4.4.2.5 SUCHE

Dieses Programm findet im Text das Vorkommen eines eingegebenen Stichwortes. Dazu werden mehrere Zeiger benutzt. So läuft einer im String, bis der angezeigte Buchstabe gleich dem ersten Buchstaben im Stichwort ist. Trifft dieser Fall ein, so werden 2 weitere Zeiger aktiviert. Einer davon läuft dann im Stichwort, der andere im String an der Stelle, wo das Stichwort vermutet wird.

```
PROGRAMM SUCHE
   ADRESSE ist der Anfang des Editbereiches
   LIES das Stichwort
   WIEDERHOLE
   BIS String von Adresse = '§'
      WIEDERHOLE
      BIS dieser String zu Ende
         WIEDERHOLE
         BIS der Buchstabe nicht zum Stichwort passt
            WENN Stichwortlänge = 0
               DANN
                  DRUCKE String aus
               SONST erhöhe Zeiger im String
                  und Zeiger im Stichwort
                  erniedrige die Stichwortlänge
            ENDE
         ENDE
         ERHOEHE Zeiger im String
      ENDE
      NAECHSTER STRING
   ENDE
ENDE
```

4.4.3 Das Programm

```
DECLARE
   VAR DISTANZ VAR TEXTADR
   ARRAY GRUNDADRESSE 7
   STRING § '§'
   VAR WORT VAR ADR
   VAR H1 VAR H2 VAR H3 VAR H4
   VAR H5 VAR H6 VAR H7
   VAR ALT VAR EGABE VAR EADR VAR EADRN
```

```
   VAR ANF VAR EN VAR SWORT   VAR SWZG
   VAR STRNGZG1 VAR STRNGZG2
   VAR SWLNG VAR STRLNG
   STRING H1 'DRUCKE'
   STRING H2 'LISTE'
   STRING H3 'EINGABE'
   STRING H4 'LOESCHE'
   STRING H5 'SUCHE'
   STRING H6 'AKTIONEN'
   STRING H7 'ENDE'
DEND

PROGRAM EDBEREICH
   SYSFIELD ?
END

PROGRAM ADRESSE
   GRUNDADRESSE DISTANZ ? 2 * +
END

PROGRAM STERN
   11 0 FOR
      WRITE '*'
        LOOP
END

PROGRAM STRNGADR
   ? ADRESSE :=
   DISTANZ ? INC DISTANZ :=
END

PROGRAM UEBSCHRIFT
   STERN
   PRINT(S)
   STERN CR
END

PROGRAM $AKTION
   CR STERN STERN STERN CR
   WRITE 'FOLGENDE AKTIONEN SIND DEFINIERT'
   CR
   6 0 FOR
      WRITE '        '
      I DISTANZ :=
      ADRESSE PRINT(S)
      CR
       LOOP
   STERN STERN STERN CR
END

PROGRAM NEXT
   ADR ? DUP ?(B) + INC ADR :=
END
```

```
PROGRAM =§
   ADR ? § ? =(S)
END

PROGRAM $DRUCKE
   EDBEREICH ADR :=
   REPEAT
   =§ UNTIL
      ADR PRINT(S)
      CR
      NEXT
   LOOP
END

PROGRAM ZN+
   ZN ? 10 + ZN :=
END

PROGRAM ANFEND
   EDBEREICH ADR :=  10 ZN :=
   WRITE 'ERSTE ZEILENNUMMER: '
   ANF READ
   WRITE 'LETZTE ZEILENNUMMER: '
   EN READ
END

PROGRAM $LISTE
   ANFEND
   REPEAT
   =§
   ZN ? EN ? > OR UNTIL
      ZN ? ANF ? >= IF
         ZN PRINT   WRITE '     '
         ADR PRINT(S)   CR
         ZN ? ANF ? <>
         ZN ? ANF ? - 80 MOD 0= AND IF
            ALT READ
                                        THEN
                        THEN
      NEXT
      ZN+
   LOOP
END

PROGRAM $EING
   EDBEREICH ADR :=
   10 ZN :=
   WRITE 'ZEILENNUMMER ? '   ANF READ
   REPEAT
   ZN ? ANF ? >= UNTIL
      NEXT ZN+
   LOOP
   ADR ? EDBEREICH ADR :=
   REPEAT
   =§ UNTIL
```

```
      NEXT
   LOOP
   ADR ? INC DUP ALT := EADR :=
      ADR :=
   REPEAT
      TEXTADR ?
      EGABE READ(S)
      TEXTADR :=
   EGABE ? § ? =(S) UNTIL
      EGABE ? ?(B) INC EADR ? + EADRN :=
      REPEAT
      EADR ? ADR ? DEC <= UNTIL
         EADR ? ? EADRN ? :=
         EADR ? 2 - EADR :=
         EADRN ? 2 - EADRN :=
      LOOP
      EGABE ? ADR ? EGABE ? ?(B) INC
      MVBYTES
      EGABE ? ?(B) INC DUP
         ADR ? + ADR :=
         ALT ? + DUP
            ALT :=
            EADR :=
   LOOP
   8193 EADR ? DEC :=
   EADR ? DEC EDBEREICH 2 - :=
END

PROGRAM $LOESCHE
   ANFEND
   REPEAT
   =§
   ZN ? EN ? > OR UNTIL
      ZN ? ANF ? >= IF
         ADR ? DUP ?(B) + INC DUP
         ADR ? SWAP EDBEREICH 2 - ? SWAP -
         DUP ALT :=    2 +
         MVBYTES
         ADR ? ALT ? + EDBEREICH 2 - :=
                    ELSE
         NEXT
                    THEN
      ZN+
   LOOP
END

PROGRAM $SUCHE
   EDBEREICH ADR :=
   SWORT READ(S)
   REPEAT
   =§ UNTIL
      ADR ? INC STRNGZG1 :=
        ADR ? ?(B) SWORT ?
        ?(B) - STRLNG :=
      REPEAT
```

```
      STRLNG ? 0= UNTIL
         STRNGZG1 ? STRNGZG2 :=
         SWORT ? INC SWZG :=
         SWORT ? ?(B) SWLNG :=
         REPEAT
         STRNGZG2 ? ?(B)
         SWZG     ? ?(B) <> UNTIL
               STRNGZG2 ? INC STRNGZG2 :=
               SWZG     ? INC SWZG     :=
               SWLNG    ? DEC SWLNG    :=
            SWLNG ? 0= IF
               ADR PRINT(S) CR
                        THEN
         LOOP
         STRLNG ? DEC STRLNG :=
         STRNGZG1 ? INC STRNGZG1 :=
      LOOP
      NEXT
   LOOP
END

PROGRAM EDITOR
   H1 STRNGADR
   H2 STRNGADR
   H3 STRNGADR
   H4 STRNGADR
   H5 STRNGADR
   H6 STRNGADR
   H7 STRNGADR
   SYSFIELD ? TEXTADR :=
   SYSFIELD ? 80 + SYSFIELD :=
   EDBEREICH EDBEREICH 2 - :=
   EDBEREICH ALT :=
   8193 EDBEREICH :=
   0 DISTANZ :=
   STERN
   WRITE ' AKTIONENLISTE '
   STERN CR CR
   REPEAT
      CR CR
      WRITE 'BITTE GEBEN SIE IHRE AKTIONEN EIN :    '
      WORT READ(S)
      WORT ? H7 ? =(S) IF
         WRITE 'ENDE DES PROGRAMMES'
                        ELSE
         WORT ? H1 ?  =(S) IF
            H1    UEBSCHRIFT $DRUCKE
                           ELSE
            WORT ? H2 ?  =(S) IF
               H2    UEBSCHRIFT $LISTE
                              ELSE
               WORT ? H3 ?  =(S) IF
                  H3    UEBSCHRIFT $EING
                                 ELSE
                  WORT ? H4 ?  =(S) IF
```

```
                    H4    UEBSCHRIFT $LOESCHE
                                ELSE
                    WORT ? H5 ?  =(S) IF
                       H5    UEBSCHRIFT $SUCHE
                                   ELSE
                       WORT ? H6 ?  =(S) IF
                          H6    UEBSCHRIFT $AKTION
                                       ELSE
                              WRITE 'DIE AKTION ' WORT PRINT(S)
                              WRITE ' IST NICHT DEFINIERT'
                              $AKTION
      THEN THEN THEN THEN
    THEN THEN THEN
   WORT ? H7 ? =(S) UNTIL
   LOOP
END
```

5.1 Das Assemblerlisting

```
; *
        PAGE    61
        .Z80
        TITLE   RPNL-COMPILER   Version 1.0
; *
; *
; *
; *
; *       ORIGINAL:       27.06.1983
; *       MODIFIED:
; *
; *
; *
; *
; *       COPYRIGHT (C) BY
; *       DIPL.-ING. GUSTAV WOSTRACK
; *
; *
; ****************************************
; *
; *
; *
WARMST          EQU     0000H
LOGIO           EQU     0005H
FCB             EQU     005CH
START           EQU     0100H
CMPEND          EQU     0C000H
; *
; *
                ASEG
                ORG     START
; *
; *
; *
; *               Initialisierung und Laufzeitsystem
; *               ----------------------------------
;
; *               Beim  Start des Programmes erfolgt eine Anpassung
; *               des  Compilers  an  das  jeweils  gerade benutzte
; *               CP/M-System.  Da  insbesondere  alle Ausgaben des
; *               Compilers  und  die  von Anwenderprogrammen nicht
; *               die entsprechenden BDOS - Funktionen, sondern die
; *               wesentlich  schnelleren  BIOS Routinen benutzten,
; *               müssen  die  aktuellen  Startadressen  dieser
; *               Routinen  aus  der  BIOS - Sprungleiste entnommen
; *               und  in  die  entsprechen  Stellen  im  Compiler
; *               eingetragen  werden.  Nachdem  der  Stackpointer
; *               gesetzt  ist,  wird  der  Interpretations-Pointer
; *               (IP)  mit  der  Adresse der ersten auszuführenden
```

```
; *             Routine geladen. Durch den anschließenden Sprung
; *             zum Laufzeitsystem NEXT wird dann diese Routine
; *             gestartet. Die Funktion von NEXT, CODE und CEND
; *             ist ausführlich im Textteil erklärt.
; *             Die angesprochene erste Routine bringt lediglich
; *             eine Meldung der Betriebsbereitschaft des
; *             Compilers. Anschließend gerät der Compiler in
; *             eine Schleife, die das eigentliche Dialogsystem
; *             ermöglicht, also den Aufruf von Programmen.
; *             Dazu wird das Prozedur INTERACT gestartet. Nach
; *             einem erfolgreichen Programmablauf oder nach
; *             einer entsprechenden Fehlermeldung kehrt das
; *             System immer wieder in genau diese Schleife
; *             zurück.
; *
                LD      A,(0004)
                LD      (ACTDRV),A
                CALL    MVLSTKRN        ; VERSCHIEBE DIE LISTEN
NSTART:         LD      HL,(WARMST+1)
                LD      DE,0009
                ADD     HL,DE           ; HL := CONSOLE OUTPUT
                LD      (OUTTBX+1),HL
                LD      (OUTTBY+1),HL
                LD      (DRUCK+2),HL
                LD      HL,ZCNT
                LD      (HL),0
                LD      SP,PSTACK
STRTCP:         LD      HL,NEWSTRT-1
                LD      (IP),HL
                JP      NEXT
; *
; *
ACTDRV:         DEFB    00
; *
; *
NEWSTRT:        DEFW    CTITLE
INTLOOP:        DEFW    EX,PEEKW,EXEC
                DEFW    JUMP,INTLOOP-1
EX:             DEFW    VARIABLE,INTERACT
                DEFS    50H
PSTACK          EQU     $
IP:             DEFW    0000
; *             ZEIGT AUF DEN NÄCHSTEN NAMEN-1
RSTACK:         DEFW    $+2
                DEFS    50H
; *
; *
NEXT:           ; Interner Übersetzer
; *
                LD      HL,(IP)
                INC     HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                LD      (IP),HL
```

```
                EX      DE,HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                EX      DE,HL
                JP      (HL)
; *
; *
CODE:           ; Beginn des
                ; zusammengesetzten Typs
; *
                LD      HL,(IP)
                EX      DE,HL
                LD      (IP),HL
                LD      HL,(RSTACK)
                LD      (HL),E
                INC     HL
                LD      (HL),D
                INC     HL
                LD      (RSTACK),HL
                JP      NEXT
; *
; *
CEND:           ; Ende des
                ; zusammengesetzten Typs
; *
                DEFW    $+2
                LD      HL,(RSTACK)
                DEC     HL
                LD      D,(HL)
                DEC     HL
                LD      E,(HL)
                LD      (RSTACK),HL
                EX      DE,HL
                LD      (IP),HL
                JP      NEXT
; *
; *
INC:
; *             Diese  Routinen  laden das oberste Wort vom Stack
; *             in das Doppelregister HL, erhöhen bzw erniedrigen
; *             dessen  Wert um eins und legen diesen veränderten
; *             Wert wieder auf den Stack.
; *
                DEFW    $+2
                POP     HL
                INC     HL
                PUSH    HL
                JR      NEXT
; *
; *
DEC:
; *
                DEFW    $+2
                POP     HL
```

```
                DEC     HL
                PUSH    HL
                JP      NEXT
; *
; *
PEEKB:          ; lädt ein Byte
; *
; *             Der  Wert  von TOP wird als Adresse interpretiert
; *             und  das  unter  dieser Adresse abgelegte Byte in
; *             das  Register  DE  geladen. Das höherwertige Byte
; *             wird auf Null gesetzt.
; *
                DEFW    $+2
                POP     HL
                LD      E,(HL)
                LD      D,00
                PUSH    DE
                JP      NEXT
; *
; *
PEEKW:          ; lädt zwei Byte
; *
; *             Der  Wert  von TOP wird als Adresse interpretiert
; *             und  die  unter  dieser und der folgenden Adresse
; *             abgelegten  Bytes  werden  in  das  Register  DE
; *             geladen.
; *
                DEFW    $+2
                POP     HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                PUSH    DE
                JP      NEXT
; *
; *
POKEB:          ; legt ein Byte ab
; *
; *             Der  Wert  von TOP wird als Adresse interpretiert
; *             und  das  niederwertige Byte des Wertes von TOP-1
; *             unter dieser Adresse abgelegt.
; *
                DEFW    $+2
                POP     HL
                POP     DE
                LD      (HL),E
                JP      NEXT
; *
; *
POKEW:          ; legt zwei Byte ab
; *
; *             Der  Wert  von TOP wird als Adresse interpretiert
; *             und  der  Wert  von  TOP-1  unter  dieser und der
; *             folgenden Adresse abgelegt.
; *
```

```
                DEFW    $+2
                POP     HL
                POP     DE
                LD      (HL),E
                INC     HL
                LD      (HL),D
                JP      NEXT
; *
; *
PUSH:           ; legt das nächste Wort
                ; auf den Stack
; *
                DEFW    $+2
                LD      HL,(IP)
                INC     HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                PUSH    DE
                LD      (IP),HL
                JP      NEXT
; *
; *
POP:            ; holt ein Wort vom Stack
; *
                DEFW    $+2
                POP     HL
                JP      NEXT
; *
; *
SWAP:           ; vertauscht TOP und TOP-1
; *
                DEFW    $+2
                POP     HL
                EX      (SP),HL
                PUSH    HL
                JP      NEXT
; *
; *
DUP:            ; dupliziert TOP nach TOP-1
; *
                DEFW    $+2
                POP     HL
                PUSH    HL
                PUSH    HL
                JP      NEXT
; *
; *
JUMP:           ; unbedingter Sprung
; *             Der Inhalt der Speicherstellen auf die der IP
; *             zeigt, wird in das Doppelregister DE geladen und
; *             dann wieder auf IP abgelegt. Dadurch erfolgt die
; *             Programmfortsetzung an einer Adresse, die hinter
; *             dem Aufruf dieser Routine steht.
; *             Beispiel:
```

```
; *             JUMP XXXX ===> Programmfortsetzung bei XXXX
; *
                DEFW    $+2
JUMP1:          LD      HL,(IP)
                INC     HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                EX      DE,HL
                LD      (IP),HL
                JP      NEXT
; *
; *
IFZ:            ; bedingter Sprung
; *     IFZ,IFNZ,IFEQ,IFGT
; *             Alle  diese  Routinen  arbeiten  nach  dem  o.a.
; *             Prinzip.  Allerdings mit dem Unterschied, daß der
; *             Sprung  nur  unter  gewissen  Voraussetzungen
; *             (Erfüllung  einer  Bedingung,  z.B. TOP ist Null)
; *             durchgeführt  wird.  Andernfalls  erfolgt  die
; *             Programmfortsetzung bei der Adresse IFXX + 4.
; *
                DEFW    $+2
                POP     HL
                LD      A,H
                OR      L
                JR      Z,JUMP1
SKIP:           LD      HL,(IP)
                INC     HL
                INC     HL
                LD      (IP),HL
                JP      NEXT
; *
; *
IFNZ:           ; bedingter Sprung
; *
                DEFW    $+2
                POP     HL
                LD      A,H
                OR      L
                JR      NZ,JUMP1
                JR      SKIP
; *
; *
IFEQ:           ; bedingter Sprung
; *
                DEFW    $+2
                POP     HL
                POP     DE
                AND     A
                SBC     HL,DE
                JR      Z,JUMP1
                JR      SKIP
; *
; *
```

```
IFGT:           ; bedingter Sprung
; *
                DEFW    $+2
                POP     HL
                POP     DE
                AND     A
                SBC     HL,DE
                JR      C,JUMP1
                JR      SKIP
; *
; *
CNSTNT:
; *
                EX      DE,HL
                INC     HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                PUSH    DE
                JP      NEXT
; *
; *
VARIABLE:
; *
                INC     DE
                PUSH    DE
                JP      NEXT
; *
; *
ZERO:           DEFW    CNSTNT
                DEFW    0000
; *
; *
ONE:            DEFW    CNSTNT
                DEFW    0001
; *
; *
MIN1:           DEFW    CNSTNT
                DEFW    0FFFFH
; *
; *
EXEC:           ; führt ein Programm aus
; *
                DEFW    $+2
                POP     HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                EX      DE,HL
                JP      (HL)
; *
; *
MATCH:          ; vergleicht zwei Worte
; *
; *     ENTRY:  TOP     -       Zeiger auf das Wort
```

```
; *                TOP-1   -       Zeiger auf das andere Wort
; *      EXIT:     TOP     -       -1 (TRUE)  wenn =
; *                                0  (FALSE) wenn ≠
; *                TOP-1   -       Zeiger auf das erste Wort
; *                TOP-2   -       Zeiger auf das andere Wort
; *
                  DEFW    $+2
                  POP     HL
                  POP     DE
                  PUSH    DE
                  PUSH    HL
MATCHO:           LD      A,(DE)
                  CP      (HL)
                  JR      NZ,MATCHF
                  LD      B,A
MATCH1:           INC     HL
                  INC     DE
                  LD      A,(DE)
                  CP      (HL)
                  JR      NZ,MATCHF
                  DJNZ    MATCH1
                  LD      HL,OFFFFH
                  PUSH    HL
                  JP      NEXT
MATCHF:           LD      HL,0000
                  PUSH    HL
                  JP      NEXT
; *
; *
COUT:             ; Zeichenausgabe
; *
                  DEFW    $+2
                  POP     BC
                  CALL    OUTPUT
                  JP      NEXT
; *
; *
CIN:              ; Zeicheneingabe
; *
                  DEFW    $+2
                  CALL    INPUT
                  LD      L,A
                  LD      H,00
                  PUSH    HL
                  JP      NEXT
; *
; *
DRON:
; *      DRON/DRUCK/DROFF
; *                Wird die Prozedur DRON aufgerufen, so wird in die
; *                Ausgaberoutine OUTTAB an 2 definierten Stellen
; *                die Adresse der BIOS-Sprungleiste ausgetauscht
; *                mit der Adresse der Routine DRUCK. Dies bedeutet,
; *                daß nun bei der Ausgabe eines Zeichens nicht mehr
; *                direkt, sondern erst innerhalb von DRUCK die
```

```
; *             entsprechende Ausgaberoutine im BIOS angesprungen
; *             wird.  Zusätzlich kann aber nun eine Ausgabe über
; *             den  Drucker  erfolgen. Bei Aufruf von DROFF wird
; *             die     alte     Adressenkonfiguration     wieder
; *             hergestellt.
; *
                DEFW    $+2
                LD      HL,DRUCK
                LD      (OUTTBX+1),HL
                LD      (OUTTBY+1),HL
                JP      NEXT
; *
; *
DRUCK:
                PUSH    BC
                CALL    0000
                POP     BC
                LD      E,C
                LD      C,05H
                JP      BTRBSYS
; *
; *
DROFF:
                DEFW    $+2
                LD      HL,(WARMST+1)
                LD      DE,0009
                ADD     HL,DE           ; HL := CONSOLE OUTPUT
                LD      (OUTTBX+1),HL
                LD      (OUTTBY+1),HL
                JP      NEXT
; *
; *
OUTPUT:
; *
; *             Ausgaberoutine OUTPUT/OUTTAB
; *             Bei  jeder Ausgabe werden alle Register gerettet.
; *             Dies  ist  notwendig,  um  einmal  eine  schnelle
; *             Stringausgabe, bei der die Anzahl der Zeichen mit
; *             übergeben  wird, zu ermöglichen, aber auch um die
; *             Umschaltung   auf   den  Drucker  durchführen  zu
; *             können.  Insbesondere  ist  dies  aber  für  die
; *             Realisierung   der  Tabulatorfunktion  notwendig.
; *             Immer  dann,  wenn  die Ausgaberoutine OUTTAB zur
; *             Ausgabe des Zeichens 09 aufgefordert wird, werden
; *             soviele  Leerzeichen  eingefügt  bis  die nächste
; *             durch  8  teilbare  Cursorposition  erreicht ist.
; *             Dazu existiert der Zähler ZCNT, der die Länge der
; *             aktuellen Zeile mitzählt.
; *
                PUSH    BC
                PUSH    DE
                PUSH    HL
OUTPT1:         CALL    OUTTAB
                POP     HL
                POP     DE
```

```
                POP     BC
OUTPT2:         RET
                ; Sprung zur Ausgaberoutine
                LD      A,C
OUTPT3:         JP      DISCOUT
; *
; *
OUTTAB:         ; AUSGABE MIT TABULATOR ÜBER DEN BIOS
                ; ZEICHEN IN C
                ; ZCNT IST EIN ZÄHLER, DER BEI OD UND OA AUF
                ; NULL GESETZT WIRD
                LD      HL,ZCNT
                LD      A,C
                CP      09H
                JR      NZ,NOTAB
                LD      C,20H
                LD      A,(HL)
                LD      B,(HL)
                ADD     A,08H
                AND     0F8H
                LD      (HL),A
OUTTB2:         CP      B
                RET     Z
                INC     B
                PUSH    BC
                PUSH    AF
OUTTBX:         CALL    0000
                POP     AF
                POP     BC
                JR      OUTTB2
NOTAB:          CP      0DH
                JR      NZ,OUTTB3
ZCNTO:          LD      (HL),0
                JR      OUTTBY
OUTTB3:         CP      0AH
               JR      Z,ZCNTO
                INC     (HL)
OUTTBY:         CALL    0000
                RET
; *
ZCNT:           DEFB    00
; *
; *
INPUT:
; *
                LD      C,01
INPUT1:         JP      BTRBSYS
                ; Sprung zum Betriebssystem
INPUT2:         JP      DISCOUT
; *
; *
RETURN:         ; wenn keine Ausgabe gewünscht
; *
                RET
; *
```

```
; *
SCROFF:         ; schaltet die Ausgabe aus
; *
; *             Wird dagegen garkeine Ausgabe über den Bildschirm
; *             gewünscht, so wird der Aufruf von OUTTAB
; *             innerhalb OUTPUT dadurch unterbunden, daß dort
; *             die Adresse der Routine RETURN, die sich wirklich
; *             lediglich aus einem RÜcksprungbefehl
; *             zusammensetzt, eingetragen wird. Die Folge ist
; *             natürlich, daß die Ausgaberoutinen überhaupt
; *             nicht erreicht werden. Dieser Effekt ist
; *             insbesondere im Zusammenwirken mit der Routine
; *             TWRITEF interessant, die nämlich den Abschluß von
; *             OUTPUT dadurch verzögert, daß der
; *             Rücksprungbefehl überschrieben und die Routine
; *             DISCOUT angesprungen wird, die dann alle
; *             auszugebenden Zeichen auf der Diskette ablegt.
; *
                DEFW    $+2
                LD      HL,RETURN
                LD      (OUTPT1+1),HL
                JP      NEXT
; *
; *
SCRON:          ; schaltet die Ausgabe ein
; *
                DEFW    $+2
                LD      HL,OUTTAB
                LD      (OUTPT1+1),HL
                JP      NEXT
; *
; *
BTRBSYS:        ; sicherheitshalber
; *
; *             Dies stellt die Schnittstelle zum CP/M
; *             Betriebssystem dar. Denn hinter LOGIO verbirgt
; *             sich die Adresse 0005.
; *
                PUSH    HL
                PUSH    DE
                PUSH    BC
                CALL    LOGIO
                POP     BC
                POP     DE
                POP     HL
                RET
; *
; *
WRTOUT:         ; Ausgabe für Texte, die durch
                ; WRITE eingegeben wurden
; *
; *             Diese Routine gibt die Strings aus, die im
; *             Compiliervorgang mit WRITE eingegeben wurden.
; *             Zunächst wird die Adresse berechnet, an der das
; *             Programm nach der Ausgabe fortgesetzt wird. Diese
```

```
; *             ergibt  sich  aus der Adresse des Strings auf die
; *             die  Länge des Strings addiert wurde. Diese Summe
; *             wird  dann  unter  IP  abgelegt.  Damit  ist  die
; *             Fortsetzung  des Programmes an dieser Stelle beim
; *             nächsten   Aufruf  von  NEXT  garantiert.  Vorher
; *             allerdings  wird  noch der Text in einer Schleife
; *             ausgegeben.
; *
                DEFW    $+2
                LD      HL,(IP)
                INC     HL      ; ZEIGT AUF DEN TEXT
                PUSH    HL
                LD      C,(HL)
                LD      B,00
                ADD     HL,BC
                LD      (IP),HL ; DORT GEHTS WEITER
                                ; NACH DEM TEXT
                POP     HL
                LD      B,C
WRT1:           INC     HL
                LD      C,(HL)
                CALL    OUTPUT
                DJNZ    WRT1
                JP      NEXT
; *
; *
ASCBIN:         ; verwandelt ASCII-Zeichen
                ; in Binärzahl
; *
; *             Konvertierungs- und Rechenroutinen
; *             Diese  Programmteile  sind  bekannt  und anderswo
; *             schon oft genug beschrieben.
; *     ENTRY:  TOP     -       Zeiger auf ein Wort
; *             TOP-1   -       ---
; *     EXIT:   TOP     -       -1 wenn konvertiert werden konnte
; *                             0  wenn nicht
; *             TOP-1   -       Binärzahl, wenn konvertiert wurde
; *                             Zeiger auf das Wort, wenn nicht
; *
                DEFW    $+2
                POP     DE
                PUSH    DE
                LD      A,(DE)
                LD      B,A
                LD      HL,0000
ASC1:           INC     DE
                LD      A,(DE)
                CP      30H
                JR      C,ASC2
                CP      3AH
                JR      NC,ASC2
                SUB     30H
                PUSH    DE
                ADD     HL,HL
                PUSH    HL
```

```
                ADD     HL,HL
                ADD     HL,HL
                POP     DE
                ADD     HL,DE
                LD      E,A
                LD      D,00
                ADD     HL,DE
                POP     DE
                DJNZ    ASC1
                POP     DE
                PUSH    HL
                LD      HL,0FFFFH
                PUSH    HL
                JP      NEXT
ASC2:           LD      HL,0000
                PUSH    HL
                JP      NEXT
; *
; *
BINASC:         ; verwandelt Binärzahl
                ; in ASCII-Zeichen
; *
; *     ENTRY:  TOP     -       16 BIT positive Integerzahl
; *             TOP-1   -       ---
; *     EXIT:   TOP     -       Zeiger auf den konvertierten
; *                             ASCII-String
; *             TOP-1   -       ---
; *
                DEFW    $+2
                LD      HL,NUM+6
                LD      B,06
CON1:           LD      (HL),00
                DEC     HL
                DJNZ    CON1
                POP     HL
                XOR     A
                LD      D,A
                LD      B,10H
CON2:           ADD     HL,HL
                ADC     A,A
                DAA
                LD      E,A
                LD      A,D
                ADC     A,A
                DAA
                LD      D,A
                RL      C
                LD      A,E
                DJNZ    CON2
                LD      HL,NUM+6
                LD      (HL),E
                DEC     HL
                LD      (HL),D
                DEC     HL
                LD      (HL),C
```

```
                LD      DE,NUM+1
                SUB     A
                LD      (DE),A
                INC     DE
                RRD
                LD      (DE),A
                INC     DE
                INC     HL
                RLD
                LD      (DE),A
                INC     DE
                RLD
                LD      (DE),A
                INC     DE
                INC     HL
                RLD
                LD      (DE),A
                RLD
                LD      (HL),A
                LD      HL,NUM+1
                LD      B,05
CON3:           LD      A,(HL)
                ADD     A,20H
                CP      20H
                JR      NZ,CON5
                LD      (HL),A
                INC     HL
                DJNZ    CON3
CON5:           INC     B
CON4:           LD      A,(HL)
                OR      30H
                LD      (HL),A
                INC     HL
                DJNZ    CON4
                LD      HL,NUM
                PUSH    HL
                JP      NEXT
; *
; *
NUM:            ; diese Zahl wird als
                ; STRING ausgedruckt
; *
                DEFB    06,0,0,0,0,0,0
; *
; *
ADD:
; *
                DEFW    $+2
                POP     HL
                POP     DE
                ADD     HL,DE
                PUSH    HL
                JP      NEXT
; *
; *
```

```
SUB:
; *
                DEFW    $+2
                POP     DE
                POP     HL
                AND     A
                SBC     HL,DE
                PUSH    HL
                JP      NEXT
; *
; *
DIV:
; *
                DEFW    CODE
                DEFW    DIVMOD,POP
                DEFW    CEND
; *
; *
MOD:
; *
                DEFW    CODE
                DEFW    DIVMOD,SWAP,POP
                DEFW    CEND
; *
; *
DIVMOD:
; *
                DEFW    $+2
                POP     BC
                POP     DE
                LD      HL,0000
                DEC     BC
                LD      A,B
                CPL
                LD      B,A
                LD      A,C
                CPL
                LD      C,A
                LD      A,10H
DIV2:           ADD     HL,HL
                PUSH    AF
                EX      DE,HL
                ADD     HL,HL
                EX      DE,HL
                JR      NC,DIV3
                INC     L
DIV3:           POP     AF
                JR      C,DIV5
                PUSH    HL
                ADD     HL,BC
                JR      C,DIV4
                POP     HL
                JR      DIV6
DIV4:           INC     E
                INC     SP
```

```
                INC     SP
                JR      DIV6
DIV5:           ADD     HL,BC
                INC     E
DIV6:           DEC     A
                JR      NZ,DIV2
                PUSH    DE
                PUSH    HL
                JP      NEXT
; *
; *
MULT:
; *
                DEFW    $+2
                POP     HL
                POP     DE
                LD      B,10H
                LD      C,D
                LD      A,E
                EX      DE,HL
                LD      HL,0000
MULT1:          SRL     C
                RRA
                JR      NC,MULT2
                ADD     HL,DE
MULT2:          EX      DE,HL
                ADD     HL,HL
                EX      DE,HL
                DJNZ    MULT1
                PUSH    HL
                JP      NEXT
; *
; *
MVBYTES:        ; transportiert Bytes
; *
                DEFW    $+2
                POP     BC
                POP     DE
                POP     HL
                LDIR
                JP      NEXT
; *
; *
RND:
; *
                DEFW    CODE
                DEFW    ZUF1,PEEKW
                DEFW    PUSH,0031,MULT
                DEFW    PUSH,1870,MOD
                DEFW    PUSH,0001,ADD
                DEFW    ZUF1,POKEW
                DEFW    ZUF2,PEEKW
                DEFW    PUSH,0029,MULT
                DEFW    PUSH,1729,MOD
                DEFW    PUSH,0001,ADD
```

```
                DEFW    ZUF2,POKEW
                DEFW    ZUF2,PEEKW
                DEFW    PUSH,0005,DIV
                DEFW    ZUF1,PEEKW
                DEFW    PUSH,0013,DIV
                DEFW    MULT
                DEFW    CEND
; *
ZUF1:           DEFW    VARIABLE
                DEFW    0123
ZUF2:           DEFW    VARIABLE
                DEFW    0234
; *
; *
FORINIT:
; *
; *               Realisierung der FOR-LOOP Schleife
; *               Siehe unter Compilerroutinen
                DEFW    CODE
                DEFW    SWAP,INC,FPUSH2
                DEFW    DEC,FPUSH2
                DEFW    CEND
; *
; *
FORTEST:
; *
                DEFW    $+2
                CALL    FPOP
                INC     DE
                PUSH    DE
                CALL    FPOP
                POP     HL
                PUSH    HL
                AND     A
                SBC     HL,DE
                JR      Z,ENDFOR
                CALL    FPUSH1
                POP     DE
                CALL    FPUSH1
                LD      HL,0000
                PUSH    HL
                JP      NEXT
ENDFOR:         POP     HL
                LD      HL,0FFFFH
                PUSH    HL
                JP      NEXT
; *
; *
I:
; *
                DEFW    $+2
                LD      DE,0FFFFH
                LD      HL,(FSTACK)
                ADD     HL,DE
                LD      D,(HL)
```

```
                DEC     HL
                LD      E,(HL)
                PUSH    DE
                JP      NEXT
; *
; *
FPUSH1:
; *
                LD      HL,(FSTACK)
                LD      (HL),E
                INC     HL
                LD      (HL),D
                INC     HL
                LD      (FSTACK),HL
                RET
; *
; *
FPUSH2:
; *
                DEFW    $+2
                POP     DE
                CALL    FPUSH1
                JP      NEXT
; *
; *
FPOP:
; *
                LD      HL,(FSTACK)
                DEC     HL
                LD      D,(HL)
                DEC     HL
                LD      E,(HL)
                LD      (FSTACK),HL
                RET
; *
; *
FSTACK:
; *
                DEFW    $+2
                DEFS    20H
; *
; *
CR:
; *
                DEFW    $+2
                LD      C,0DH
                CALL    OUTPUT
                LD      C,0AH
                CALL    OUTPUT
                JP      NEXT
; *
; *
PRINT:
; *
                DEFW    CODE
```

```
                DEFW    PEEKW,BINASC,PRINTS
                DEFW    CEND
; *
; *
PRINTS:         ; druckt STRING
; *
; *     ENTRY:  TOP     -       Zeiger auf den String
; *             TOP-1   -       ---
; *     EXIT:   TOP     -       ---
; *             TOP-1   -       ---
; *
                DEFW    $+2
                POP     HL
                LD      B,(HL)
PRNTS1:         INC     HL
                LD      C,(HL)
                CALL    OUTPUT
                DJNZ    PRNTS1
                JP      NEXT
; *
; *
PRINT.S.:
; *
                DEFW    CODE
                DEFW    PEEKW,PRINTS
                DEFW    CEND
; *
; *
READ.N.:        ; Eingabe fÜr Zahlen
; *             Eingaberoutinen für Strings (READ.N./READ.S./
                                                        READ.CH.)
; *             Alle  diese  Routinen bauen auf einer gemeinsamen
; *             Basis auf, die beschrieben werden soll.
; *             Unter  einer  Adresse,  die unter SYSFIELD+2
; *             erreichbar  ist,  wird  zunächst  ein  Byte
; *             reserviert, in das abschließend die Länge des
; *             dann  eingegebenen  Strings geschrieben wird.
; *             Bekanntlich ist dies dann der 'head' des Strings.
; *             Bei  der  Eingabe  eines  Zeichens  wird  jeweils
; *             geprüft,  ob  es  sich  um ein BACKSPACE handelt,
; *             dann  nämlich  wird  die Länge korrigiert und der
; *             Zeiger  (Doppelregister DE)  auf  das  aktuelle
; *             Zeichen  zurückgesetzt,  oder  ob  ein  Carriage
; *             Return  eingegeben  wurde.  Dann  nämlich ist die
; *             Eingabe  beendet  und  das  Programm verzweigt zu
; *             NEXT.  Innerhalb  des  Programmes werden Optionen
; *             behandelt. So können durch eventuelle Subtraktion
; *             von 20H aus Klein- Großbuchstaben werden.
; *
                DEFW    $+2
                LD      A,20H
                LD      (RCH2+5),A
                LD      DE,(SYSFIELD+2)
                SUB     A
                LD      (DE),A
```

```
                JR      RCH10
; *
; *
READ.S.:        ; liest Zeichen ein
; *
                DEFW    $+2
                LD      A,00
                JR      RCH0
; *
; *
READ.CH.:       ; liest Zeichen ein
; *                 Kleinbuchstaben werden zu großen
                DEFW    $+2
                LD      A,20H
RCH0:           LD      (RCH2+5),A
                LD      DE,(SYSFIELD+2)
                POP     HL
                SUB     A
                LD      (DE),A
                LD      (HL),E
                INC     HL
                LD      (HL),D
RCH10:          PUSH    DE
                POP     HL      ; LÄNGE
RCH1:           INC     DE      ; ZEICHEN
                CALL    INPUT
                CP      08      ; BACKSPACE ?
                JR      NZ,RCH2
                DEC     DE
                LD      A,(HL)
                OR      A
                JR      Z,RCH1
                DEC     DE
                DEC     (HL)
                JR      RCH1
RCH2:           CP      60H     ; KLEINBUCHSTABE ?
                JR      C,RCH3
                SUB     20H
RCH3:           LD      (SYSFIELD+2),DE
                CP      0DH
                JP      Z,NEXT
                INC     (HL)
                LD      (DE),A
                JR      RCH1
; *
; *
INCHAR:         ; liest genau ein Zeichen
                ; und legt es unter ZEICHEN ab
; *
                DEFW    $+2
                LD      C,06
                LD      E,0FFH
INCHR1:         CALL    BTRBSYS
                OR      A
                JR      Z,INCHR1
```

```
                LD      E,A
                LD      D,00
                LD      (ZEICHEN+2),DE
                JP      NEXT
; *
ZEICHEN:
; *
                DEFW    VARIABLE
                DEFW    00
; *
; *
EQ:
; *     EQ,EQZ,$LT,$GT,$GT$EQ,$LT$EQ,$LT$GT
; *             Alle   diese   Vergleichsoperatoren  beruhen  auf
; *             demselben  Prinzip. Sie vergleichen TOP mit TOP-1
; *             und legen je nach Ergebnis ein OFFFFH für logisch
; *             'wahr' oder ein 0000 für logisch 'falsch' auf den
; *             Stack. Eine gewisse Ausnahme bildet EQZ insofern,
; *             als  der  Vergleichswert  für TOP nicht vom Stack
; *             genommen  wird,  sondern  die Konstante Null ist.
; *             Somit  ist  auch  der  Unterschied  zu  den  oben
; *             beschrieben     Prozeduren,     wie     IFZ,IFNZ...
; *             augenscheinlich. Denn bei diesen erfolgt im Fall,
; *             daß  die  gesetzte  Bedingung erfüllt ist, sofort
; *             ein  Sprung zu einem gesetzten Ziel, während hier
; *             lediglich  das  Ergebnis  der Evaluierung auf den
; *             Stack    gelegt    wird    für    eine    weitere
; *             Verarbeitung.
; *
                DEFW    $+2
                POP     HL
                POP     DE
                AND     A
                SBC     HL,DE
EQ1:            LD      HL,0000
                JR      NZ,EQ2
                DEC     HL
EQ2:            PUSH    HL
                JP      NEXT
; *
; *
EQZ:
; *
                DEFW    $+2
                POP     HL
                LD      A,H
                OR      L
                JR      EQ1
; *
; *
§EQ.S.:
; *     Vergleichsoperatoren für Strings $EQ.S.,$GEQ.S.
; *             Der erste Operator benutzt die Routine MATCH, die
; *             insbesondere  bei  der  Suche in den Namenslisten
; *             verwendet  wird.  Wie  schon  angedeutet,  werden
```

```
; *             dabei zunächst die Längen der beiden Strings
; *             verglichen und nur bei Übereinstimmung steigt man
; *             um auf byteweisen Vergleich.
; *             Der nächste Algorithmus nimmt den längeren als
; *             Bezugsstring und vergleicht solange, bis ein
; *             Abbruchkriterium erreicht ist. Und dies kann im
; *             Fall, daß die Strings gleich sind nur sein, daß
; *             sie einmal gleich lang, aber auch noch Zeichen
; *             für Zeichen identisch sind.
; *
                DEFW    $+2
                POP     HL
                POP     DE
                JP      MATCHO
; *
; *
§GEQ.S.:
; *
                DEFW    $+2
                POP     HL
                POP     DE
                LD      B,(HL)
                LD      A,(DE)
                CP      B
                JR      NC,GEQ1
                LD      B,A
GEQ1:           INC     HL
                INC     DE
                LD      A,(DE)
                LD      C,(HL)
                CP      C
                JR      Z,GEQ2
                JR      NC,GEQ3
                JR      GEQ4
GEQ2:           DJNZ    GEQ1
GEQ3:           LD      HL,0FFFFH
                PUSH    HL
                JP      NEXT
GEQ4:           LD      HL,0000
                PUSH    HL
                JP      NEXT
; *
; *
§LT:
; *
                DEFW    $+2
                POP     DE
                POP     HL
§LT1:           AND     A
                SBC     HL,DE
§LT2:           SBC     HL,HL
                PUSH    HL
                JP      NEXT
; *
; *
```

```
§GT:
; *
                DEFW    $+2
                POP     HL
                POP     DE
                JR      §LT1
; *
; *
§GT§EQ:
; *
                DEFW    $+2
                POP     DE
                POP     HL
§GTEQ1:         AND     A
                SBC     HL,DE
                CCF
                JR      §LT2
; *
; *
§LT§EQ:
; *
                DEFW    $+2
                POP     HL
                POP     DE
                JR      §GTEQ1
; *
; *
§LT§GT:
; *
                DEFW    CODE
                DEFW    EQ,NOT
                DEFW    CEND
; *
; *
AND:
; *     AND,OR,NOT
; *             Diese    Operatoren    werden    in    der    vorne
; *             beschriebenen Weise auf die obersten Elemente des
; *             Stack  angewandt. Das Ergebnis wird ebenfalls auf
; *             den Stack abgelegt.
; *
                DEFW    $+2
                POP     HL
                POP     DE
                LD      A,H
                AND     D
                LD      H,A
                LD      L,A
                PUSH    HL
                JP      NEXT
; *
; *
OR:
; *
                DEFW    $+2
```

```
                POP     HL
                POP     DE
                LD      A,H
                OR      D
                LD      H,A
                LD      L,A
                PUSH    HL
                JP      NEXT
; *
; *
NOT:
; *
                DEFW    $+2
                POP     HL
                ADD     HL,HL
                CCF
                SBC     HL,HL
                PUSH    HL
                JP      NEXT
; *
; *
TRUE:
; *
                DEFW    $+2
                LD      HL,(MIN1+2)
                PUSH    HL
                JP      NEXT
; *
; *
FALSE:
; *
                DEFW    $+2
                LD      HL,(ZERO+2)
                PUSH    HL
                JP      NEXT
; *
; *
EXIT:
; *
; *             Diese   Funktion   wird  beim  Ausstieg  aus  dem
; *             Compiler und Rückkehr zu CP/M aufgerufen. Vor der
; *             eigentlichen  Rückkehr  wird  der DMA-Bereich auf
; *             die Voreinstellung von 80H gesetzt.
; *
                DEFW    $+2
                LD      DE,0080H
                LD      C,1AH
                CALL    BTRBSYS
                JP      WARMST
; *
; *
                ; Beginn des Kodebereiches
; *
SYSFIELD:       DEFW    VARIABLE
                DEFW    FIELD
```

```
; *
; *
OPEN:
; *
; *          Bei  Start  dieser Routine liegt die Basisadresse
; *          des  zugehörigen  DMA-Bereiches  auf  dem  Stack.
; *          Durch Addition der DMA-Bereichlänge wird zunächst
; *          ein   Zeiger  erzeugt,  der  anfänglich  auf  den
; *          sogenannten  DMA-Pointer weist, der dann auch auf
; *          Null  gesetzt  wird.  Der  DMA-Pointer  gibt  die
; *          aktuelle  Stelle  innerhalb des DMA-Bereiches an,
; *          in die nun geschrieben bzw aus der gelesen werden
; *          kann. In diesem Zusammenhang sei auf die Prozedur
; *          WRITEWF  hingewiesen.  Anschließend  werden  noch
; *          Zähler im FCB zurückgesetzt und dann dann endlich
; *          wird  die Datei mit diesem FCB geöffnet. Wird auf
; *          der  aktuellen  Diskette  kein  passender Eintrag
; *          gefunden,  so  wird  einer  erzeugt. Ist dies aus
; *          Platzgründen  nicht  möglich,  dann  erfolgt eine
; *          Fehlermeldung.
; *
                DEFW    $+2
                POP     HL
                LD      DE,0080H
                ADD     HL,DE
                LD      A,00
                LD      (HL),A
                INC     HL
                PUSH    HL
                LD      DE,000CH
                ADD     HL,DE
                XOR     A
                LD      B,04
OPEN1:          LD      (HL),A
                INC     HL
                DJNZ    OPEN1
                LD      DE,0010H
                ADD     HL,DE
                LD      (HL),A
                POP     DE
                LD      C,0FH
                CALL    BTRBSYS
                CP      0FFH     ; KEINE GEFUNDEN
                JP      NZ,NEXT
                LD      C,16H
                CALL    BTRBSYS
                CP      0FFH
                JP      NZ,NEXT
                LD      C,09H
                LD      DE,DISKFULL
                CALL    BTRBSYS
                JP      NEXT
; *
DISKFULL:       DEFB    0DH,0AH,'DISK FULL $'
; *
```

```
FCB2:           DEFS    80H
                DEFB    0       ; DMAPOINTER
                DEFB    0,20H,20H,20H
                DEFB    20H,20H,20H,20H
                DEFB    20H,'RPN',0,0,0,0
                DEFB    0,20H,20H,20H
                DEFB    20H,20H,20H,20H
                DEFB    20H,20H,20H,20H,0,0,0,0
                DEFB    0,0,0,0
; *
; *
CLOSE:
; *
; *               Diese Prozedur erzeugt einen Zeiger auf den
; *               zugehörigen FCB und ruft die entsprechende
; *               BDOS-Funktion auf. Dadurch wird der FCB auf die
; *               Diskette geschrieben und die Datei dauerhaft
; *               gesichert. Darüberhinaushinaus wird der
; *               DMA-Pointer auf Null gesetzt.
; *
                DEFW    $+2
                LD      C,10H
                POP     HL
                LD      DE,0080H
                ADD     HL,DE
                LD      A,00
                LD      (HL),A
                INC     HL
                EX      DE,HL
                CALL    BTRBSYS
                JP      NEXT
; *
; *
DELETE:         ; lösche den Eintrag
; *
; *               Diese Prozedur erzeugt einen Zeiger auf den
; *               zugehörigen FCB und ruft die entsprechende
; *               BDOS-Funktion auf. Dadurch werden alle Dateien
; *               auf der aktuellen Diskette gelöscht, auf die der
; *               angezeigte FCB paßt.
; *
                DEFW    $+2
                POP     HL
                LD      DE,0081H
                ADD     HL,DE
                EX      DE,HL
                LD      C,13H
                CALL    BTRBSYS
                JP      NEXT
; *
; *
READF:
; *
; *               Diese Prozedur geht davon aus, daß auf dem Stack
; *               die Adresse des zugehörigen DMA-Breiches liegt.
```

```
; *             Diese wird dann mit der BDOS-Funktion 'SET DMA
; *             ADDRESS' dem CP/M mitgeteilt. Damit wird
; *             erreicht, daß beim nachfolgenden Lesevorgang die
; *             eingelesen Daten auch dorthin geschrieben werden.
; *             Abschließend wird mit Hilfe des vom System zur
; *             Verfügung gestellten Suchkodes ein boolscher
; *             Parameter auf den Stack gelegt, der angibt, ob
; *             der Lesevorgang erfolgreich war oder nicht.
; *
                DEFW    $+2
                POP     DE
                PUSH    DE
                LD      C,1AH
                CALL    BTRBSYS
                LD      C,14H
                POP     DE
                LD      HL,0081H
                ADD     HL,DE
                EX      DE,HL
                CALL    BTRBSYS
                OR      A
                JR      Z,READOK
                LD      HL,0000
                PUSH    HL
                JP      NEXT
READOK:         LD      HL,0FFFFH
                PUSH    HL
                JP      NEXT
; *
; *
TWRITEF:
; *
; *             Die Prozedur selbst setzt lediglich einige
; *             Parameter, wie den Anfang des DMA-Bereiches und
; *             einen Zeiger auf den DMA-Pointer. Insbesondere
; *             aber wird durch Manipulationen an den Aus- und
; *             Eingabeprozeduren erreicht, daß jedes Zeichen,
; *             das aus- oder eingegeben wird, an die Prozedur
; *             DISCOUT übergeben wird.
; *             Dort wird, nachdem alle Register gerettet sind,
; *             durch Addition des DMA-Pointers (ACTF2) auf die
; *             Basisadresse des DMA-Bereiches ein Zeiger auf den
; *             aktuellen Speicherplatz erzeugt, wohin auch das
; *             übergebene Zeichen geschrieben wird. Anschließend
; *             wird überprüft, ob der inkrementierte DMA-Pointer
; *             schon gleich 80H, das ist die Gesamtlänge des
; *             DMA-Bereiches, ist. Ist dies nicht der Fall, so
; *             wird ein EOF-Zeichen (1AH) angehängt. Ist
; *             allerdings der DMA-Bereich voll, so werden diese
; *             128 Byte als sequentielle Datei auf Diskette
; *             geschrieben und der DMA-Bereich durch eine Null
; *             im DMA-Pointer und dem EOF-Zeichen zu Beginn des
; *             Bereiches als leer gekennzeichnet.
; *
                DEFW    $+2
```

```
                POP     HL
                LD      (ACTF1),HL
                LD      DE,0080H
                ADD     HL,DE
                LD      (ACTF2),HL
                LD      A,00H
                LD      (OUTPT2),A
                LD      A,0CDH
                LD      (INPUT1),A
                LD      HL,DISCOUT
                LD      (OUTPT3+1),HL
                LD      (INPUT2+1),HL
                JP      NEXT
; *
; *
DISCOUT:
; *
                PUSH    AF
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      HL,(ACTF2)
                LD      E,(HL)
                LD      D,00
                LD      HL,(ACTF1)
                ADD     HL,DE
                LD      (HL),A
                LD      A,E
                INC     A
                CP      80H
                INC     HL
                LD      (HL),1AH
                JR      NZ,DSC1
                LD      DE,(ACTF1)
                LD      C,1AH
                CALL    BTRBSYS
                LD      DE,(ACTF2)
                INC     DE
                LD      C,15H
                CALL    BTRBSYS
                LD      HL,(ACTF1)
                LD      (HL),1AH
                LD      A,00
DSC1:           LD      HL,(ACTF2)
                LD      (HL),A
                POP     HL
                POP     DE
                POP     BC
                POP     AF
                RET
; *
ACTF1:          DEFW    0000    ; AKT. DATEI
; *
; *
ACTF2:          DEFW    0000    ; AKT. DMAPNT
```

```
; *
; *
TREADF:
; *
                DEFW    $+2
                POP     HL          ; ZEIGER AUF DMA-BEREICH
                LD      (ACTF3),HL
                LD      DE,0080H
                ADD     HL,DE
                LD      (ACTF4),HL          ; DMA-POINTER
                LD      DE,(SYSFIELD+2)
                POP     HL          ; ADRESSE DES STRING
                SUB     A           ; LÄNGE NULL ZUNÄCHST
                LD      (DE),A
                LD      (HL),E
                INC     HL
                LD      (HL),D
                PUSH    DE          ; DE: ZEIGER AUF DAS ZEICHEN
                POP     HL          ; HL: ZEIGER AUF DIE LÄNGE
TRCH1:          INC     DE
                LD      (SYSFIELD+2),DE
                LD      (ACTLNG),HL
                LD      HL,(ACTF4)
                LD      E,(HL)  ; DMA POINTER
                LD      D,00
                LD      HL,(ACTF3)
                ADD     HL,DE
                LD      A,E
                INC     E
                INC     A
                CP      80H
                LD      A,(HL)
                LD      BC,0FFFFH
                JR      NZ,TDSC1
                LD      BC,0000
                LD      E,00
TDSC1:          LD      HL,(ACTF4)
                LD      (HL),E
                CP      1AH                 ; DATEIENDE
                JR      NZ,TRCH2
                LD      HL,(ACTF4)
                LD      (HL),00
                LD      BC,0000
                PUSH    BC
                PUSH    BC
                JP      NEXT
TRCH2:          PUSH    BC
                LD      C,A
                CALL    OUTPUT
                LD      A,C
                POP     BC
                CP      0AH
                JR      Z,TRCH4
                CP      0DH
                JR      NZ,TRCH3
```

```
                LD      HL,(ACTLNG)
                SUB     A         ; DER STRING BESTEHT
                CP      (HL)      ; NUR AUS EINEM CR
                JR      NZ,TRCH21
                INC     (HL)      ; DANN LEGE EIN BLANK AB
                LD      A,' '
                LD      DE,(SYSFIELD+2)
                LD      (DE),A
                INC     DE
                LD      (SYSFIELD+2),DE
TRCH21:         LD      HL,(ACTF4)
                INC     (HL)
                LD      A,(HL)
                CP      01H
                JR      Z,TRCH20
                CP      80H
                JR      NZ,TRCH22
                LD      (HL),00
TRCH20:         LD      BC,0FFFFH         ; DMABEREICH-ENDE UND
                PUSH    BC                ; SATZENDE
                LD      BC,0000
                JR      TRCH23
TRCH22:         LD      BC,0FFFFH         ; DMABEREICH NICHT ZU EN-
                PUSH    BC                ; DE, ABER SATZENDE
TRCH23:         PUSH    BC
                LD      C,0AH
                CALL    OUTPUT
                JP      NEXT
TRCH3:          LD      HL,(ACTLNG)
                LD      DE,(SYSFIELD+2)
                INC     (HL)
                LD      (DE),A
TRCH4:          PUSH    BC
                LD      A,B
                OR      A
                JP      NZ,TRCH5
                LD      BC,0FFFFH         ; DMABEREICH-ENDE
                PUSH    BC                ; ABER KEIN SATZENDE
                LD      DE,(SYSFIELD+2)
                INC     DE
                LD      (SYSFIELD+2),DE
                JP      NEXT
TRCH5:          POP     BC
                JP      TRCH1
; *
ACTF3:          DEFW    0000      ; AKT. DATEI
; *
; *
ACTF4:          DEFW    0000      ; AKT. DMAPNT
; *
; *
ACTLNG:         DEFW    0000
; *
; *
TEND:
```

```
; *
; *             Diese Prozedur setzt die mit TWRITEF angebrachten
; *             Änderungen am Ein- Ausgabesystem wieder zurück.
; *             Zunächst wird hinter der normalen Ausgabe wieder
; *             ein Rücksprungbefehl angebracht, so daß die
; *             Prozedur DISCOUT nicht mehr erreicht wird.
; *             Während im TWRITEF-Modus die Eingabe als
; *             Unterprogramm aufgerufen und anschließend zu
; *             DISCOUT verzweigt wird, wird durch TEND die
; *             Eingabe wieder durch einen Sprungbefehl erreicht
; *             und damit DISCOUT abgeschnitten.
; *
                DEFW    $+2
                LD      A,0C9H
                LD      (OUTPT2),A
                LD      A,0C3H
                LD      (INPUT1),A
                JP      NEXT
; *
; *
WRITEF:
; *
; *             Ähnlich wie die Leseprozedur setzt die
; *             Schreibprozedur den aktuellen DMA-Bereich fest
; *             und schreibt dann sequentiell auf die Diskette.
; *             Dabei sind wie bei allen BDOS-Aufrufen die
; *             geforderten Parameter, hier ist es ein Zeiger auf
; *             den FCB im Doppelregister DE, zu übergeben.
; *
                DEFW    $+2
                POP     DE
                PUSH    DE
                LD      C,1AH
                CALL    BTRBSYS
                LD      C,15H
                POP     DE
                LD      HL,0081H
                ADD     HL,DE
                EX      DE,HL
                CALL    BTRBSYS
                JP      NEXT
; *
; *
PRINTF:
; *
; *             Diese Prozedur erwartet auf dem Stack einen
; *             Zeiger auf einen DMA-Bereich. Dieser Bereich,
; *             also 128 Byte werden auf dem Ausgabemedium
; *             dargestellt, falls nicht zwischenzeitlich ein EOF
; *             (1AH) erkannt wird.
; *
                DEFW    $+2
                POP     HL
                LD      B,80H
PRNTF1:         LD      A,(HL)
```

```
                INC     HL
                CP      1AH
                JP      Z,NEXT
                LD      C,A
                CALL    OUTPUT
                DJNZ    PRNTF1
                JP      NEXT
; *
; *
GETDIR:
; *
; *             Nach    Ausgabe   des   Textes,   der   als   DATSTRNG
; *             deklariert  ist,  wird  mit  dem  FCB  DFCB,  der
; *             sämtlich  aus  Fragezeichen  besteht,  der  erste
; *             passende   Eintrag  im  Directory  der  aktuellen
; *             Diskette  gesucht.  Bekanntlich paßt zu dem 'wild
; *             card'  ?  jedes  Zeichen,  so  daß das System mit
; *             einem  Zeiger auf den ersten Eintrag im Directory
; *             antwortet.  Aus  diesem  Zeiger  kann  dann  die
; *             Adresse  des  zugehörigen  FCB  im  vorgesetzten
; *             DMA-Bereich ab 80H berechnet und anschließend der
; *             Dateiname  ausgegeben werden. Danach wird mit der
; *             Funktion   SEARCH   NEXT   in   gleicher   Weise
; *             weitergesucht und auch alle gefundenen Dateinamen
; *             ausgegeben.
; *
                DEFW    $+2
                LD      DE,0080H
                LD      C,1AH
                CALL    BTRBSYS
                LD      A,00
                LD      (DIRCNT),A
                LD      DE,DATSTRNG
                LD      C,09H   ; AUSGABE STRING
                CALL    BTRBSYS
                LD      A,(ACTDRV)
                INC     A
                LD      (DFCB),A
                ADD     A,40H
                LD      C,A
                CALL    OUTPUT
                LD      C,':'
                CALL    OUTPUT
                LD      DE,NEWL
                LD      C,09H
                CALL    BTRBSYS
                LD      DE,DFCB
                LD      C,11H
                CALL    BTRBSYS ; SEARCH FIRST
GDIR2:          CALL    DIRPRNT
                LD      A,(DIRCNT)
                INC     A
                CP      04
                JR      NZ,GDIR3
                LD      DE,NEWL
```

```
                LD      C,09H
                CALL    BTRBSYS
                LD      A,00
GDIR3:          LD      (DIRCNT),A
                LD      DE,DFCB
                LD      C,12H
                CALL    BTRBSYS ; SEARCH NEXT
                CP      0FFH
                JP      Z,NEXT
                JR      GDIR2
; *
DIRCNT:         DEFB    0
; *
NEWL:           DEFB    0DH,0AH,'       $'
; *
DIRPRNT:        LD      L,A
                LD      H,00
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,HL
                LD      DE,0080H
                ADD     HL,DE
                LD      B,8
DPRNT1:         INC     HL
                LD      C,(HL)
                CALL    OUTPUT  ; DATEINAME
                DJNZ    DPRNT1
                LD      C,'.'
                CALL    OUTPUT  ; .
; *
                LD      B,03
DPRNT2:         INC     HL
                LD      C,(HL)
                CALL    OUTPUT  ; DATEITYP
                DJNZ    DPRNT2
                LD      B,06H
                LD      C,' '
DPRNT3:         CALL    OUTPUT
                DJNZ    DPRNT3
                RET
; *
DFCB:           DEFB    0,'?','?','?'
                DEFB    '?','?','?','?'
                DEFB    '?','?','?','?',0,0,0,0
; *
DATSTRNG:       DEFB    '     Dateien auf dem Laufwerk $'
; *
; *
CDRIVE:
; *
; *               Unter   der  Speicherstelle  ACTDRV  (im  Listing
; *               ziemlich  am Anfang) steht der Kode des aktuellen
; *               Laufwerkes.  Wird  dazu  die Zahl 41H addiert, so
```

```
; *             entsteht daraus das entsprechende ASCII-Zeichen
; *             und kann dann mit einem Text (DRMSG1) ausgegeben
; *             werden. Anschließend wird zur Eingabe des
; *             gewünschten Betriebslaufwerkes aufgefordert.
; *             Dabei wird untersucht, ob es sich bei dem
; *             eingegebenen Zeichen um einen Kleinbuchstaben
; *             handelt (`-60H), denn dann muß anders subtrahiert
; *             werden, als dies bei Großbuchstaben der Fall ist,
; *             um zum geforderten Laufwerkkode zu gelangen.
; *
                DEFW    $+2
                LD      DE,DRMSG1
                LD      C,09
                CALL    BTRBSYS
                LD      A,(ACTDRV)
                ADD     A,41H
                LD      C,A
                CALL    OUTPUT
CDRV2:          LD      DE,DRMSG2
                LD      C,09
                CALL    BTRBSYS
                CALL    INPUT
                CP      60H
                JR      C,CDRV3
                SUB     20H
CDRV3:          SUB     41H
                JP      M,CDRV2
                CP      08
                JR      NC,CDRV2
                LD      (ACTDRV),A
                JP      NEXT
; *
DRMSG1:         DEFB    0DH,0AH,'     Aktuelles   Laufwerk: $'
DRMSG2:         DEFB    0DH,0AH,'     Gewünschtes Laufwerk: $'
; *
; *
RDFNAME:
; *
; *             Bei Start dieser Prozedur liegt auf dem Stack ein
; *             Zeiger auf den zugehörigen FCB. Dort wird
; *             anschließend nach den CP/M-Vorschriften ein
; *             Dateiname eingetragen.
; *
                DEFW    $+2
                POP     HL
RDFN1:          PUSH    HL
                LD      DE,0082H
                ADD     HL,DE    ; HL=FILENAME
                LD      B,09
RDFN2:          CALL    IPTX
                CP      1BH      ; ESCAPE
                JR      NZ,RDFN15
                POP     HL
                LD      DE,0082H
                ADD     HL,DE
```

```
                LD      B,0BH
RDFN14:         LD      (HL),' '
                INC     HL
                DJNZ    RDFN14
                JP      NEXT
RDFN15:         CP      '.'
                JR      Z,RDFN3
                CP      60H
                JR      C,RDFN20
                SUB     20H
RDFN20:         CP      08
                JR      NZ,RDFN21
                LD      A,B       ; BACKSPACE
                CP      09
                JR      Z,RDFN2
                LD      C,08      ; AUSGABE FÜR BACKSPACE
                CALL    OUTPUT
                INC     B
                DEC     HL
                LD      (HL),' '
                JR      RDFN2
RDFN21:         LD      (HL),A
                INC     HL
                LD      C,A       ; AUSGABE FÜR NORM. ZEICHEN
                CALL    OUTPUT
                DJNZ    RDFN2
                LD      DE,FNMSG
                LD      C,09
                CALL    BTRBSYS
                POP     HL
                JR      RDFN1
RDFN3:          LD      C,A
                CALL    OUTPUT    ; AUSGABE PUNKT
                DEC     B
RDFN4:          JR      Z,RDFN5
                LD      A,' '
                LD      (HL),A
                INC     HL
                DJNZ    RDFN4
RDFN5:          LD      B,03
RDFN6:          CALL    IPTX
                CP      1BH       ; ESCAPE : NEUE EINGABE
                JR      NZ,RDFN70
                POP     HL
                JR      RDFN1
RDFN70:         CP      60H
                JR      C,RDFN60
                SUB     20H
RDFN60:         CP      08
                JR      NZ,RDFN61
                LD      A,B
                CP      3
                JR      Z,RDFN6
                LD      C,08
                CALL    OUTPUT
```

```
                INC     B
                DEC     HL
                LD      (HL),' '
                JR      RDFN6
RDFN61:         LD      (HL),A
                INC     HL
                LD      C,A
                CALL    OUTPUT
                DJNZ    RDFN6
                POP     HL
                JP      NEXT
; *
FNMSG:          DEFB    0DH,'     Ungültiger Name              $'
; *
IPTX:
; *
                LD      E,0FFH
                LD      C,06H   ; DIRECT CONSOLE INPUT
                CALL    BTRBSYS
                CP      21H
                JR      C,IPTX2 ; SPRINGE BEI SONDERZEICHEN
                RET
IPTX2:          CP      08      ; BACKSPACE
                RET     Z
                CP      2EH     ; PUNKT
                RET     Z
                CP      1BH     ; ESCAPE
                RET     Z
                JR      IPTX
; *
; *
NOFILE:
; *     SCHALTET ALLE FILE-AKTIONEN AUS
; *
; *             Bekanntlich ist das Ende der Namensliste durch
; *             eine Null, d.h. einen Eintrag mit der Länge Null
; *             gekennzeichnet.  Normalerweise steht dieses
; *             Zeichen auch am Ende der Liste. NOFILE trägt nun
; *             an der Stelle LST2 eine Null ein, wodurch die
; *             Suche in der Liste hier abgebrochen wird. Der
; *             Kode sämtlicher Routinen, die dadurch nicht mehr
; *             ereichbar sind, steht hinter dem der Prozedur
; *             OPEN. Da es sich bei diesem Kode nun um 'garbage'
; *             handelt, wird die Systemvariable SYSFIELD, die
; *             den Beginn des Kodebereiches festgelegt, mit
; *             dieser Adresse geladen.
; *
                DEFW    $+2
                LD      HL,OPEN
                LD      (SYSFIELD+2),HL
                LD      A,00
                LD      (LST2),A
                JP      NEXT
; *
```

```
; *
FIELD           EQU     $
; *
; *             hier beginnt der
; *             Variablenbereich
; *             und anschließend der Kodebereich
; *
; *
; *
                .PHASE CMPEND
; *
; *
; *             Alles, was jetzt kommt, wird beim Start an das
; *             Ende des Speichers kopiert und anschließend von
; *             compilierten Programmen überschrieben. Es handelt
; *             sich bei diesen Routinen um Programme aus der
; *             Compilerliste, die lediglich während der
; *             Compilezeit benötigt werden. Bei Ausführung des
; *             SAVE-Befehles wird dieser Programmteil nicht
; *             mitabgespeichert, da er bei der Laufzeit nicht
; *             benötigt wird. Durch diese Trennung wird einmal
; *             der Compiler gleichzeitig aber auch die
; *             abgespeicherten Anwenderprogramme kürzer.`A´
; *
; *
; *             hier beginnt die Namensliste
; *             des Benutzers
; *
; *     NAMENSLISTE
; *
NAMES:          DEFB    04,'CODE'
                DEFW    TCODE
                DEFB    04,'CALL'
                DEFW    TCALL
                DEFB    01,'?'
                DEFW    PEEKW
                DEFB    04,'?(B)'
                DEFW    PEEKB
                DEFB    02,':='
                DEFW    POKEW
                DEFB    05,':=(B)'
                DEFW    POKEB
                DEFB    03,'INC'
                DEFW    INC
                DEFB    03,'DUP'
                DEFW    DUP
                DEFB    03,'DEC'
                DEFW    DEC
                DEFB    01,'+'
                DEFW    ADD
                DEFB    01,'-'
                DEFW    SUB
                DEFB    03,'DIV'
                DEFW    DIV
                DEFB    01,'*'
```

```
DEFW    MULT
DEFB    03,'MOD'
DEFW    MOD
DEFB    02,'CR'
DEFW    CR
DEFB    03,'NOT'
DEFW    NOT
DEFB    03,'AND'
DEFW    AND
DEFB    02,'OR'
DEFW    OR
DEFB    07,'PROGRAM'
DEFW    PROGRAM
DEFB    06,'OUTPUT'
DEFW    OUTPUT
DEFB    01,'='
DEFW    EQ
DEFB    02,'0='
DEFW    EQZ
DEFB    01,'<'
DEFW    §LT
DEFB    01,'>'
DEFW    §GT
DEFB    02,'>='
DEFW    §GT§EQ
DEFB    02,'<='
DEFW    §LT§EQ
DEFB    02,'<>'
DEFW    §LT§GT
DEFB    01,'I'
DEFW    I
DEFB    04,'SWAP'
DEFW    SWAP
DEFB    04,'=(S)'
DEFW    §EQ.S.
DEFB    05,'>=(S)'
DEFW    §GEQ.S.
DEFB    08,'PRINT(S)'
DEFW    PRINT.S.
DEFB    07,'READ(S)'
DEFW    READ.S.
DEFB    08,'READ(CH)'
DEFW    READ.CH.
DEFB    06,'INCHAR'
DEFW    INCHAR
DEFB    07,'OUTCHAR'
DEFW    COUT
DEFB    05,'PRINT'
DEFW    PRINT
DEFB    03,'NEW'
DEFW    NEW
DEFB    04,'TRUE'
DEFW    TRUE
DEFB    05,'FALSE'
DEFW    FALSE
```

```
                DEFB    07,'DECLARE'
                DEFW    DECLARE
                DEFB    07,'COMPILE'
                DEFW    COMPILE
                DEFB    06,'SCROFF'
                DEFW    SCROFF
                DEFB    05,'SCRON'
                DEFW    SCRON
                DEFB    06,'SYSTEM'
                DEFW    EXIT
                DEFB    08,'SYSFIELD'
                DEFW    SYSFIELD
                DEFB    07,'BTRBSYS'
                DEFW    BTRBSYS
                DEFB    07,'ZEICHEN'
                DEFW    ZEICHEN
                DEFB    04,'DRON'
                DEFW    DRON
                DEFB    05,'DROFF'
                DEFW    DROFF
                DEFB    04,'LIST'
                DEFW    LIST
                DEFB    06,'MEMORY'
                DEFW    MEMORY
                DEFB    03,'RND'
                DEFW    RND
                DEFB    04,'SAVE'
                DEFW    SAVE
LST2:
; *             Die folgenden Befehle können
; *             durch NOFILES ausgeschaltet werden
; *
                DEFB    05,'FILES'
                DEFW    GETDIR
                DEFB    08,'CHGDRIVE'
                DEFW    CDRIVE
                DEFB    09,'DATEINAME'
                DEFW    RDFNAME
                DEFB    05,'READF'
                DEFW    READF
                DEFB    04,'OPEN'
                DEFW    OPEN
                DEFB    05,'CLOSE'
                DEFW    CLOSE
                DEFB    06,'DELETE'
                DEFW    DELETE
                DEFB    06,'WRITEF'
                DEFW    WRITEF
                DEFB    06,'PRINTF'
                DEFW    PRINTF
                DEFB    07,'TWRITEF'
                DEFW    TWRITEF
                DEFB    06,'TREADF'
                DEFW    TREADF
                DEFB    04,'TEND'
```

```
                DEFW    TEND
                DEFB    06,'ACTDRV'
                DEFW    ACTDRV
                DEFB    07,'NOFILES'
                DEFW    NOFILES
                DEFB    00
; *
; *     ENDE DER
; *     NAMENSLISTE
; *
; *
; *     COMPILERLISTE
; *
CDICT:          DEFB    04,'CEND'
                DEFW    TCEND
                DEFB    02,'IF'
                DEFW    IF
                DEFB    04,'THEN'
                DEFW    THEN
                DEFB    04,'ELSE'
                DEFW    ELSE
                DEFB    06,'REPEAT'
                DEFW    REPEAT
                DEFB    05,'UNTIL'
                DEFW    UNTIL
                DEFB    04,'LOOP'
                DEFW    LOOP
                DEFB    03,'FOR'
                DEFW    FOR
                DEFB    03,'END'
                DEFW    END
                DEFB    05,'WRITE'
                DEFW    WRITE
                DEFB    04,'READ'
                DEFW    READ
                DEFB    00
; *
; *             Diese Schlüsselworte sind
; *             in einer Deklaration erlaubt
; *
DECLST:         DEFB    03,'VAR'
                DEFW    VAR
                DEFB    04,'VARF'
                DEFW    VARF
                DEFB    05,'CONST'
                DEFW    CONST
                DEFB    05,'ARRAY'
                DEFW    ARRAY
                DEFB    06,'STRING'
                DEFW    STRING
                DEFB    04,'DEND'
                DEFW    END
                DEFB    00
; *
; *     ENDE DER COMPILERLISTE
```

```
; *
; *
; *             KERN 2
; *
; *
CTITLE:         DEFW    CODE
                DEFW    PUSH,STRTMS
                DEFW    PRINTS
                DEFW    CEND
STRTMS:         DEFB    93H
                DEFB    0DH,0AH
        DEFB    '>>   RPNL - COMPILER          VERS. 1.0    <<'
        DEFB    0DH,0AH
        DEFB    '>>   Copyright    Dipl.-Ing. G. Wostrack    <<'
        DEFB    0DH,0AH
        DEFB    '>>                                          <<'
; *
; *
INIT:
; *
                DEFW    CODE
                DEFW    ZERO,CFLAG,POKEW
                DEFW    GETWORD,ENTER
                DEFW    CEND
; *
; *
EINGABE:
; *
; *             Dieser    Algorithmus    ist    dem    später    zu
; *             beschreibenden PROGRAM ähnlich. Deshalb sei an
; *             dieser  Stelle  lediglich  auf  den  Unterschied
; *             hingewiesen,   der   darin   besteht,   daß   alle
; *             Zahleneingaben in sedezimaler Form bzw deren
; *             Repräsentation  als  ASCII-Zeichen  möglich sind.
; *             Die   Aufgabe   der   Konvertierung   zur   binären
; *             Darstellung übernimmt das Programm HEX.
; *
                DEFW    CODE
EIN1:           DEFW    GETWORD,HEX
                DEFW    IFZ,EIN2-1
                DEFW    CMPLW
                DEFW    SYSFIELD,PEEKW,DEC
                DEFW    SYSFIELD,POKEW
                DEFW    JUMP,EIN5-1
EIN2:           DEFW    SYSNAMES,PEEKW,LOOKUP
                DEFW    IFZ,EIN3-1
                DEFW    CMPLW
                DEFW    JUMP,EIN5-1
EIN3:           DEFW    SYSCDICT,PEEKW,LOOKUP
                DEFW    IFZ,EIN4-1,EXEC
                DEFW    JUMP,EIN5-1
EIN4:           DEFW    SCRON,PRINTS
                DEFW    PUSH,COMPMSSG
                DEFW    ERR1
EIN5:           DEFW    CFLAG,PEEKW
```

```
                DEFW    IFNZ,EIN6-1
                DEFW    JUMP,EIN1-1
EIN6:           DEFW    CEND
; *
; *
HEX:
; *
                DEFW    $+2
                POP     DE
                PUSH    DE
                LD      A,(DE)
                LD      B,02
                CP      B
                JR      NZ,HEX3
                XOR     A
                LD      HL,ZSPEICHER
                LD      (HL),A
                INC     HL
                LD      (HL),A
                INC     DE
HEX1:           LD      A,(DE)
                DEC     HL
                SUB     30H
                JP      M,HEX3
                CP      0AH
                JR      C,HEX2
                SUB     07H
                CP      0AH
                JP      M,HEX3
                CP      10H
                JP      P,HEX3
HEX2:           INC     DE
                RLD
                INC     HL
                RLD
                DJNZ    HEX1
                POP     HL
                LD      HL,(ZSPEICHER)
                PUSH    HL
                LD      HL,(MIN1+2)
                JR      HEX4
HEX3:           LD      HL,(ZERO+2)
HEX4:           PUSH    HL
                JP      NEXT
ZSPEICHER:      DEFW    0000
; *
; *
TCEND:
; *
; *             Dieser  Vorgang  läuft solange, bis EINGABE einen
; *             Fehler  entdeckt,  oder  aber  die  zum  dem
; *             Schlüsselwort  CEND  gehörige  Routine  TCEND das
; *             Compilerflag zurücksetzt (auf TRUE).
; *
                DEFW    CODE
```

```
                DEFW    MIN1,CFLAG,POKEW
                DEFW    CEND
; *
; *
TCALL:
; *
; *             Durch   diese   Routine   wird   die   Eingabe   von
; *             Sedezimalzahlen  in  das  Programm ermöglicht. Da
; *             dieser Vorgang einem Compilieren entspricht, wird
; *             zunächst  durch die Routine INIT das Compilerflag
; *             auf  FALSE  gesetzt,  der zugehörige Programmname
; *             gelesen   (GETWORD)   und   in   die   Namensliste
; *             eingetragen   (ENTER).  Anschließend  werden mit
; *             EINGABE    die    Daten    in    den    Kodebereich
; *             übertragen.
; *
                DEFW    CODE
                DEFW    INIT
                DEFW    EINGABE
                DEFW    CEND
; *
; *
TCODE:
; *
; *             Dies ist die dem Schlüsselwort CODE entsprechende
; *             Routine,   die   bekanntlich   den   Programmtyp   2
; *             erzeugt. Dadurch unterscheidet sie sich von TCALL
; *             dadurch,  daß vor dem Parameterfeld das Kodefeld,
; *             das  aus  einem  Zeiger  auf dieses Parameterfeld
; *             besteht,  einzufügen  ist. Dazu wird die aktuelle
; *             Programmadresse von SYSFIELD geladen, um 2 erhöht
; *             und  dann auf (SYSFIELD) wieder abgelegt. Mit der
; *             Routine  KORRSYS  wird ein Sprungbefehl nach NEXT
; *             an   das   Kernprogramm   angehangen   und   SYSFIELD
; *             aktualisiert.
; *
                DEFW    CODE
                DEFW    INIT
                DEFW    SYSFIELD,PEEKW,DUP
                DEFW    INC,INC,DUP
                DEFW    SYSFIELD,POKEW,SWAP
                DEFW    POKEW
                DEFW    EINGABE,KORRSYS
                DEFW    CEND
; *
; *
KORRSYS:
; *
                DEFW    $+2
                LD      HL,(SYSFIELD+2)
                LD      A,0C3H
                LD      (HL),A
                LD      DE,NEXT
                INC     HL
                LD      (HL),E
```

```
                INC     HL
                LD      (HL),D
                INC     HL
                LD      (SYSFIELD+2),HL
                JP      NEXT
; *
; *
NEW:
; *
                DEFW    $+2
                LD      A,0C9H
                LD      (OUTPT2),A
                LD      A,0C3H
                LD      (INPUT1),A
                LD      HL,BTRBSYS
                LD      (INPUT1+1),HL
                LD      HL,NAMES
                LD      (SYSNAMES+2),HL
                LD      HL,FIELD
                LD      (SYSFIELD+2),HL
                JP      NSTART
; *
; *
COMPILE:
; *
; *               Die COMPILE-Funktion, die das Compilieren von
; *               Quell-Programmen aus Dateien auf der Diskette
; *               ermöglicht, tauscht im wesentlichen lediglich die
; *               Eingaberoutine, die die Tastatur abfragt, gegen
; *               eine Eingabe aus, die die Zeichen aus dem
; *               DMA-Puffer bezieht.
; *               Bei Aufruf des Schlüsselwortes COMPILE wird
; *               zunächst ein FCB mit dem gewünschten Namen
; *               eingerichtet. Anschließend wird zu diesem FCB
; *               passende Datei geöffnet und die Eingaberoutine
; *               zur Prozedur READFILE umgeleitet.
; *
                DEFW    CODE
                DEFW    GETWORD,MFCB1
                DEFW    CHGINP
                DEFW    CEND
; *
; *
MFCB1:
                DEFW    $+2
                LD      DE,0080H
                LD      C,1AH
                CALL    BTRBSYS
                LD      HL,FCBCOMP
                LD      DE,FCB
                LD      BC,0024H
                LDIR
                POP     HL
                LD      DE,FCB
                LD      A,(HL)
```

```
                CP      09H
                JR      C,LNGOK
                LD      A,08H
LNGOK:          LD      B,00
                LD      C,A
                INC     HL
                LD      A,(HL)
                SUB     40H
                LD      (DE),A
                INC     DE
                LDIR
                JP      NEXT
; *
; *
CHGINP:         ; CHANGE INPUT
                ; vertauscht Tastaturroutine
                ; mit Eingabe von DMA
; *
                DEFW    $+2
                LD      DE,FCB
                LD      C,0FH
                CALL    BTRBSYS
                CP      0FFH
                JR      NZ,CHINP1
                LD      DE,FLERR
                LD      C,09
                CALL    BTRBSYS
                JP      NEXT
CHINP1:         LD      HL,READFILE
                LD      (INPUT1+1),HL
                LD      A,80H
                LD      (DMAPNT),A
                JP      STRTCP
; *
DMAPNT:         DEFB    00
FLERR:          DEFB    'NO FILE $'
; *
READFILE:
; *
; *             Diese Prozedur liest nun die von CHGINP geöffnete
; *             Datei  sequentiell  und schreibt die Daten in den
; *             Standard-DMA-Puffer  (ab 80H). Ein Zeiger auf das
; *             jeweils  aktuelle  Zeichen  wird  durch  Addition
; *             eines DMA-Pointers (DMA-PNT) auf die Basisadresse
; *             des  Puffers bereitgestellt. Immer dann, wenn der
; *             Puffer  vollständig  ausgelesen  ist,  also  der
; *             Pointer  auf  80H hochgezählt ist, wird ein neuer
; *             Sektor  eingelesen. Dies geschieht genau solange,
; *             bis  das  EOF-Zeichen  (1AH)  erkannt wurde. Dann
; *             wird die Eingaberoutine wieder so umgestellt, daß
; *             Zeichen von der Tastatur angenommen werden.
; *
                LD      A,(DMAPNT)
                CP      80H
                JR      NZ,READF2
```

```
                LD      DE,FCB
                LD      C,14H
                CALL    BTRBSYS
                OR      A
                JR      Z,READF2
READF1:         LD      HL,BTRBSYS
                LD      (INPUT1+1),HL
                RET
READF2:         LD      E,A
                LD      D,00
                INC     A
                LD      (DMAPNT),A
                LD      HL,0080H
                ADD     HL,DE
                LD      C,(HL)
                CALL    OUTPUT
                LD      A,C
                CP      0EH
                JR      NZ,READF3
                LD      A,09
READF3:         CP      0AH
                JR      Z,READFILE
                CP      1AH
                RET     NZ
                LD      DE,CDGENMSG
                LD      C,09
                CALL    BTRBSYS
                LD      A,0DH
                JP      READF1
; *
CDGENMSG:       DEFB    '***CODE GENERATED***$'
; *
; *
FCBCOMP:        DEFB    0,20H,20H,20H
                DEFB    20H,20H,20H,20H
                DEFB    20H,'PRG',0,0,0,0
                DEFB    0,20H,20H,20H
                DEFB    20H,20H,20H,20H
                DEFB    20H,20H,20H,20H,0,0,0,0
                DEFB    0,0,0,0
; *
; *
READ:
; *
; *               Diese Compilerroutine generiert zur Compilezeit
; *               alle die Routinen, die während der Laufzeit des
; *               Programmes nötig werden, um die Eingabe einer
; *               Zahl zu ermöglichen.
; *               Auch eine Zahl wird zunächst als ein String
; *               eingelesen, für den ein Eingabepuffer zur
; *               Verfügung gestellt werden muß. Dies ge- schieht
; *               durch Abfrage von SYSFIELD. Unter der Adresse,
; *               die dort steht, dies ist in aller Regel hinter
; *               dem Anwenderprogramm, wird dieser String mit
; *               READ.N.  Da dieser String nur vorübergehend
```

```
; *             benötigt  wird,  wird  SYSFIELD sofort wieder auf
; *             den  ursprünglichen  Wert  zurückgesetzt. Bedingt
; *             durch  die  Syntax  liegt  bei  Eintritt  in  den
; *             Eingabemodus  die  Adresse  der Variablen auf dem
; *             Stack, der die eingegebene Zahl zugewiesen werden
; *             soll.     Nach     anschließender     erfolgreicher
; *             Konvertierung  des  Strings  in  eine 16 Bit-Zahl
; *             wird dieser Wert dort abgelegt.
; *
                DEFW    CODE
                DEFW    PUSH,SYSFIELD,CMPLW
                DEFW    PUSH,PEEKW,CMPLW
                DEFW    PUSH,DUP,CMPLW
                DEFW    PUSH,READ.N.,CMPLW
                DEFW    PUSH,SYSFIELD,CMPLW
                        ; SYSFIELD ZURÜCKSETZEN
                DEFW    PUSH,POKEW,CMPLW
                DEFW    PUSH,ASCBIN,CMPLW
                DEFW    PUSH,POP,CMPLW
                DEFW    PUSH,SWAP,CMPLW
                DEFW    PUSH,POKEW,CMPLW
                DEFW    CEND
; *
; *
GETWORD:        ; holt ein Wort von der Eingabe
; *
                DEFW    CODE
GETW1:          DEFW    SCAN
                DEFW    IFNZ,GETW2-1
                DEFW    READLINE
                DEFW    JUMP,GETW1-1
GETW2:          DEFW    CEND
; *
; *
SCAN:           ; verwaltet den Textpuffer
                ; bei Eingaben
; *
; *     ENTRY:  TOP     -       ---
; *             TOP-1   -       ---
; *     EXIT:   TOP     -       -1 wenn das Wort gefunden wurde
; *                             0  wenn nicht
; *             TOP-1   -       Zeiger auf das Wort, wenn gefunden
; *                             sonst leer
; *
; *     Beschreibung:   Das  erste Byte des Puffers enthält einen
; *             Zähler,  der  angibt,  wieviele  Zeichen  bereits
; *             verarbeitet  wurden. Das nächste Wort wird an den
; *             Anfang   geschoben   und  ein  Byte,  das  dessen
; *             aktuelle Länge angibt, vorangestellt.
; *
                DEFW    $+2
                LD      HL,BUFF+2
                LD      C,(HL)
                LD      B,00
                INC     (HL)
```

```
                JR      Z,SC5
                INC     HL
                ADD     HL,BC
                LD      B,C
SC1:            INC     HL
                INC     B
                LD      A,(HL)
                OR      A
                JR      Z,SC5
                CP      ' '
                JR      Z,SC1
                JR      SC2
                INC     HL
                INC     B
                LD      A,(HL)
SC2:            LD      DE,BUFF+3
                LD      C,00
SC3:            INC     DE
                LD      (DE),A
                INC     C
                INC     B
                INC     HL
                LD      A,(HL)
                OR      A
                JR      NZ,SC4
                LD      B,0FFH
                LD      A,' '
SC4:            CP      ' '
                JR      NZ,SC3
                LD      HL,BUFF+2
                LD      (HL),B
                INC     HL
                LD      (HL),C
                PUSH    HL
                LD      HL,0FFFFH
                PUSH    HL
                JP      NEXT
SC5:            LD      A,0FFH
                LD      (BUFF+2),A
                LD      HL,0000
                PUSH    HL
                JP      NEXT
; *
; *
READLINE:       ; füllt den Eingabepuffer
; *
; *             Diese Routine ist die Schnittstelle zum Anwender.
; *             So werden hier alle Eingaben in einem
; *             Eingabepuffer (BUFF) gesammelt und erst nach
; *             Erkennen eines Carriage Return (0DH) von der
; *             später beschriebenen Routine INTERACT
; *             ausgewertet. Darüber hinaus werden in dieser
; *             Routine alle Kommentare ausgefiltert, sodaß der
; *             eigentliche Compiler damit garnicht belastet
; *             wird.
```

```
; *
                DEFW    CODE
                DEFW    ZERO
                DEFW    BUFF,POKEB
                DEFW    BUFF,INC,INC
READL1:         DEFW    DUP
                DEFW    CIN
                DEFW    DUP,PUSH,0008
                DEFW    IFEQ,READL3-1
                DEFW    DUP,PUSH,0009
                DEFW    IFEQ,READL4-1
                DEFW    DUP,PUSH,';'
                DEFW    IFEQ,READL5-1
                DEFW    DUP,PUSH,000DH
                DEFW    IFEQ,READL6-1
READL2:         DEFW    SWAP,POKEB
                DEFW    INC
                DEFW    JUMP,READL1-1
READL3:         DEFW    POP,POP
                DEFW    DEC
                DEFW    PUSH,0020H,COUT
                DEFW    PUSH,0008H,COUT
                DEFW    JUMP,READL1-1
READL4:         DEFW    POP
                DEFW    PUSH,0020H
                DEFW    JUMP,READL2-1
READL5:         DEFW    CIN
                DEFW    PUSH,000DH
                DEFW    EQ,IFZ,READL5-1
READL6:         DEFW    POP,POP
                DEFW    ZERO,SWAP,POKEB
                DEFW    PUSH,000AH,COUT
                DEFW    CEND
; *
; *
BUFF:           ; der Eingabepuffer
; *
                DEFW    VARIABLE
                DEFS    80H
; *
; *
WRITE:
; *
; *               Diese  Compilerroutine  generiert zur Compilezeit
; *               die   Routine,   die während der Laufzeit des
; *               Programmes nötig werden, um die Ausgabe eines
; *               Strings zu ermöglichen.
; *               Dies  ist lediglich die Routine WRTOUT, die schon
; *               beschrieben  ist.  Von  MVTEXT wird dann noch der
; *               von   SCANWR   bereitgestellte   String  aus  dem
; *               Eingabepuffer   hinter   die   Routine   WRTOUT
; *               kopiert.
; *
                DEFW    CODE
                DEFW    PUSH,WRTOUT,CMPLW       ; GIBT DEN TEXT
```

```
                DEFW    SCANWR                  ; AUS
                DEFW    IFZ,COMP41-1            ; FEHLER
                DEFW    MVTEXT
                DEFW    CEND
; *
; *
MVTEXT:         ; legt Text im Variablen-
                ; bereich ab
; *     ENTRY:  TOP     -       Zeiger auf das Wort
; *             TOP-1   -       ---
; *     EXIT:   TOP     -       ---
; *
; *     Beschreibung:   Kopiert Text aus dem Puffer nach
; *                     (SYSYFIELD) und aktualisiert SYSFIELD
; *
                DEFW    $+2
                POP     HL      ; QUELLE
                LD      C,(HL)  ; ANZAHL
                LD      B,00
                INC     BC
                LD      DE,(SYSFIELD+2)
                LDIR
                EX      DE,HL
                LD      (SYSFIELD+2),HL
                JP      NEXT
; *
; *
IF:
; *
; *             Um TOP und TOP-1 auszuwerten, wird ein IFZ
; *             kompiliert. Dahinter muß die Sprungadresse
; *             geschrieben werden, die aber jetzt noch nicht
; *             bekannt ist. Deshalb wird zunächst eine Null als
; *             Platzhalter vorgesehen und die Adresse dieses
; *             Speicherplatzes auf dem Stack zur späteren
; *             Verwendung gelegt.
; *
                DEFW    CODE
                DEFW    PUSH,IFZ,CMPLW
                DEFW    SYSFIELD,PEEKW
                DEFW    ZERO,CMPLW
                DEFW    CEND
; *
; *
THEN:
; *
; *             Da dieses Schlüsselwort die Sprunganweisung
; *             abschließt erfolgt bei Nichterfüllung der
; *             Bedingung ein Sprung an diese Stelle. Somit
; *             stellt die aktuelle Adresse die in der
; *             IF-Anweisung benötigte Sprungadresse dar. Der
; *             Zeiger auf deren Platzhalter wird vom Stack
; *             genommen und die Adresse dort abgelegt.
; *
                DEFW    CODE
```

```
                DEFW    SYSFIELD,PEEKW,DEC
                DEFW    SWAP,POKEW
                DEFW    CEND
; *
; *
ELSE:
; *
; *               Die Verhältnisse sind etwas anders, wenn zwischen
; *               IF  und  THEN ein ELSE eingefügt wurde. Denn dann
; *               erfolgt  die Programmfortsetzung nicht hinter dem
; *               THEN,  sondern  hinter  dem  ELSE.  Dies bedeutet
; *               einmal,  daß  in den Platzhalter die nun aktuelle
; *               Adresse  eintragen  werden  muß,  damit bei einer
; *               Evaluierung,   die  das  Ergebnis  false  bringt,
; *               hierher gesprungen wird. Andererseits muß für den
; *               umgekehrten  Fall  vor  dem  Alternativkode  ein
; *               Sprung  zum Ende der Bedingungsanweisung (wie bei
; *               IF)  vorgesehen  werden,  der  dann  von  THEN
; *               komplettiert wird.
; *
                DEFW    CODE
                DEFW    PUSH,JUMP,CMPLW
                DEFW    SYSFIELD,PEEKW
                DEFW    ZERO,CMPLW
                DEFW    SWAP
                DEFW    SYSFIELD,PEEKW,DEC
                DEFW    SWAP,POKEW
                DEFW    CEND
; *
; *
REPEAT:
; *
; *               Diese   Wiederholungsschleife   arbeitet  nach
; *               ähnlichen  Prinzipien  wie  die  IF-ELSE-THEN
; *               Bedingung.  Denn REPEAT legt als Merker für einen
; *               eventuellen  Rücksprung  nach der Bedingung UNTIL
; *               bzw. LOOP seine Adresse auf den Stack.
; *               Das  Ausflicken  der  Löcher  im Befehlskode, das
; *               eine  zweite  Phase  erspart und auch bei anderen
; *               Compilern  Verwendung  findet,  wird  als 'fixup'
; *               bezeichnet.
; *
                DEFW    CODE
                DEFW    SYSFIELD,PEEKW,DEC
                DEFW    CEND
; *
; *
UNTIL:
; *
; *               Diese  Routine  sieht  einen Sprung vor, wenn die
; *               Bedingung  als  true  ausgewertet wird. Auch hier
; *               wird  die  Sprungadresse zunächst durch eine Null
; *               frei gehalten.
; *
                DEFW    CODE
```

```
                DEFW    PUSH,IFNZ,CMPLW
                DEFW    SYSFIELD,PEEKW
                DEFW    ZERO,CMPLW
                DEFW    CEND
; *
; *
LOOP:
; *
; *             Diese Routine generiert zunächst einen Sprung zum
; *             Anfang der Schleife mit dem dort zurückgelassenen
; *             Zeiger. Anschließend wird das bei UNTIL bzw
; *             hinter FOR vorhandene Loch mit der aktuellen
; *             Programmadresse gestopft.
; *
                DEFW    CODE
                DEFW    PUSH,JUMP,CMPLW
                DEFW    SWAP,CMPLW
                DEFW    SYSFIELD,PEEKW
                DEFW    DEC
                DEFW    SWAP,POKEW
                DEFW    CEND
; *
; *
FOR:
; *
; *             Die FOR-LOOP ist insofern etwas komplizierter, da
; *             sie zur Laufzeit die Laufparameter auswerten muß.
; *             Die Vorbereitung dazu, dies ist die
; *             Bereitstellung der Argumente in dem FSTACK, wird
; *             durch FORINIT geleistet. Die jeweilige
; *             Evaluierung der aktuellen Parameter übernimmt
; *             FORTEST. Der Aufwand, einen eigenen Stack für
; *             diese Werte zu benutzten, rührt von der
; *             Forderung, FOR-Schleifen schachteln zu dürfen.
; *             Denn damit schafft man sich die Möglichkeit,
; *             jeder tieferliegenden Schleife, die
; *             entsprechenden Werte zuordnen zu können.
; *
                DEFW    CODE
                DEFW    PUSH,FORINIT,CMPLW
                DEFW    SYSFIELD,PEEKW,DEC
                DEFW    PUSH,FORTEST,CMPLW
                DEFW    PUSH,IFNZ,CMPLW
                DEFW    SYSFIELD,PEEKW
                DEFW    ZERO,CMPLW
                DEFW    CEND
; *
; *
SCANWR:         ; verwaltet den Textpuffer
                ; bei WRITE
; *
; *             Diese Routine wird von WRITE aufgerufen, um einen
; *             String für die spätere Ausgabe zu erkennen. Dazu
; *             ist insbesondere nötig, seinen Anfang und sein
; *             Ende festzustellen (dies erkennt man an den
```

```
; *             Zeichen  '  ), um dessen Länge abzuzählen und ihm
; *             dann als head voranzustellen. Ein boolescher Wert
; *             wird  auf den Stack gelegt, der angibt, ob
; *             überhaupt ein String entdeckt wurde.
; *
                DEFW    $+2
                LD      HL,BUFF+2
                LD      C,(HL)
                LD      B,00
                INC     (HL)
                JR      Z,SCWR5
                INC     HL
                ADD     HL,BC
                LD      B,C
SCWR1:          INC     HL
                INC     B
                LD      A,(HL)
                OR      A
                JR      Z,SCWR5
                CP      27H
                JR      Z,SCWR1
                JR      SCWR2
                INC     HL
                INC     B
                LD      A,(HL)
SCWR2:          LD      DE,BUFF+3
                LD      C,00
SCWR3:          INC     DE
                LD      (DE),A
                INC     C
                INC     B
                INC     HL
                LD      A,(HL)
                OR      A
                JR      NZ,SCWR4
                LD      B,0FFH
                LD      A,27H
SCWR4:          CP      27H
                JR      NZ,SCWR3
                LD      HL,BUFF+2
                LD      (HL),B
                INC     HL
                LD      (HL),C
                PUSH    HL
                LD      HL,0FFFFH
                PUSH    HL
                JP      NEXT
SCWR5:          LD      A,0FFH
                LD      (BUFF+2),A
                LD      HL,0000
                PUSH    HL
                JP      NEXT
; *
; *
LOOKUP:         ; sucht ein Wort in den Listen
```

```
; *
; *             Dies  ist  eine  vom  Compiler  häufig  benutzte
; *             Routine.  Denn  sie  sucht nach einem Wort in den
; *             Listen.  Das  Vergleichen  der  beiden Kandidaten
; *             übernimmt  MATCH.  Das  Berechnen der Adresse des
; *             Nachfolgers  in  der  Liste wird mit der Funktion
; *             FIRST  durchgeführt, die die Länge des Kandidaten
; *             zur  Verfügung  stellt.  Bei  erfolgreicher Suche
; *             liegt  unter  dem  entsprechenden booleschen Wert
; *             die   Adresse   des  gefundenen  Namens  auf  dem
; *             Stack.
; *
                DEFW    CODE
LOOK1:          DEFW    DUP,PEEKB
                DEFW    IFZ,LOOK2-1
                DEFW    MATCH
                DEFW    IFNZ,LOOK3-1
                DEFW    FIRST,ADD
                DEFW    PUSH,0002,ADD
                DEFW    JUMP,LOOK1-1
LOOK2:          DEFW    POP
                DEFW    ZERO
                DEFW    CEND
LOOK3:          DEFW    SWAP,POP
                DEFW    FIRST,ADD,PEEKW
                DEFW    MIN1
                DEFW    CEND
; *
; *
FIRST:          ; ermittelt die Länge
                ; des STRING
; *
; *     ENTRY:  TOP     -       Zeiger auf den String
; *             TOP-1   -       ---
; *     EXIT:   TOP     -       Erstes Zeichen des String
; *                             (enthält die Länge des String)
; *             TOP-1   -       Zeiger auf den Reststring
; *
                DEFW    $+2
                POP     HL
                LD      C,(HL)
                LD      B,00
                INC     HL
                PUSH    HL
                PUSH    BC
                JP      NEXT
; *
; *
INTERACT:       ; liest eine Zeile und
                ; führt sie aus
; *
; *             Alle Anwenderprogramme werden aus diesem Programm
; *             heraus  gestartet. Der zugehörige Algorithmus ist
; *             bereits  vorne  erklärt. An dieser Stelle sei nur
; *             darauf  hingewiesen, daß bei einem Kompilat eines
```

```
; *             RPNL-Programmes  (darum  handelt  es  sich  im
; *             Assemblerlisting)   natürlich    keine Wieder-
; *             holungsschleifen  auftreten  können. Auf dieser
; *             Ebene läuft natürlich alles unstrukturiert mit
; *             gewöhnlichen Sprungbefehlen ab.
; *
                DEFW    CODE
                DEFW    PROMPT
                DEFW    READCMD
INT1:           DEFW    SCAN
                DEFW    IFZ,INT3-1
                DEFW    SYSNAMES,PEEKW
                DEFW    LOOKUP
                DEFW    IFZ,INT2-1
                DEFW    EXEC
                DEFW    JUMP,INT1-1
INT2:           DEFW    ASCBIN
                DEFW    IFNZ,INT1-1
                DEFW    PUSH,BMSSG
                DEFW    PRINTS
                DEFW    PRINTS
                DEFW    CEND
INT3:           DEFW    PUSH,OKMSSG
                DEFW    PRINTS
                DEFW    CEND
BMSS:           DEFB    0DH,'Not defined: '
OKMSSG:         DEFB    05H,0DH,0AH,'Ok!'
; *
; *
READCMD:        ; füllt den Eingabepuffer
; *                 macht Klein- Großbuchstaben
                DEFW    CODE
                DEFW    ZERO
                DEFW    BUFF,POKEB
                DEFW    BUFF,INC,INC
READC1:         DEFW    DUP
                DEFW    CCIN
                DEFW    DUP,PUSH,0008
                DEFW    IFEQ,READC3-1
                DEFW    DUP,PUSH,0009
                DEFW    IFEQ,READC4-1
                DEFW    DUP,PUSH,';'
                DEFW    IFEQ,READC5-1
                DEFW    DUP,PUSH,000DH
                DEFW    IFEQ,READC6-1
READC2:         DEFW    SWAP,POKEB
                DEFW    INC
                DEFW    JUMP,READC1-1
READC3:         DEFW    POP,POP
                DEFW    DEC
                DEFW    PUSH,0020H,COUT
                DEFW    PUSH,0008H,COUT
                DEFW    JUMP,READC1-1
READC4:         DEFW    POP
                DEFW    PUSH,0020H
```

```
                DEFW    JUMP,READC2-1
READC5:         DEFW    CIN
                DEFW    PUSH,000DH
                DEFW    EQ,IFZ,READC5-1
READC6:         DEFW    POP,POP
                DEFW    ZERO,SWAP,POKEB
                DEFW    PUSH,000AH,COUT
                DEFW    CEND
; *
; *
CCIN:
                ; konvertiert aus kleinen große
                ; Buchstaben
; *
                DEFW    $+2
                CALL    INPUT
                CP      60H
                JR      C,CCIN1
                SUB     20H
CCIN1:          LD      L,A
                LD      H,00
                PUSH    HL
                JP      NEXT
; *
; *
PROMPT:         ; druckt '#'
; *
                DEFW    CODE
                DEFW    PUSH,PMSSG
                DEFW    PRINTS
                DEFW    CEND
PMSSG:          DEFB    03H,0DH,0AH,'#'
                ; CR '#'
; *
; *
PROGRAM:        ; der Compiler
; *
; *             Dieses Schlüsselwort startet den eigentlichen
; *             Compiler. Auch dieser Algorithmus ist im Texteil
; *             ausführlich beschrieben.
; *
                DEFW    CODE
                DEFW    ZERO,CFLAG,POKEW
                DEFW    GETWORD,ENTER
                DEFW    PUSH,CODE,CMPLW
COMP1:          DEFW    GETWORD
                DEFW    SYSCDICT,PEEKW,LOOKUP
                DEFW    IFZ,COMP2-1
                DEFW    EXEC
                DEFW    JUMP,COMP5-1
COMP2:          DEFW    SYSNAMES,PEEKW,LOOKUP
                DEFW    IFZ,COMP3-1
                DEFW    CMPLW
                DEFW    JUMP,COMP5-1
COMP3:          DEFW    ASCBIN
```

```
                DEFW    IFZ,COMP4-1
                DEFW    PUSH,PUSH,CMPLW
                DEFW    CMPLW
                DEFW    JUMP,COMP5-1
COMP4:          DEFW    SCRON,PRINTS
COMP41:         DEFW    SCRON,PUSH,COMPMSSG
                DEFW    ERR1
COMP5:          DEFW    CFLAG,PEEKW
                DEFW    IFNZ,COMP6-1
                DEFW    JUMP,COMP1-1
COMP6:          DEFW    CEND
COMPMSSG:       DEFB    2AH,' is undefined.'
                DEFB    0DH,0AH,0DH,0AH
                DEFB    ' ***NO CODE GENERATED***'
; *
; *
; Systemvariablen
                ; Beginn der Namensliste
; *
SYSNAMES:       DEFW    VARIABLE
                DEFW    NAMES
; *
; *
                ; Beginn der Compilerliste
; *
SYSCDICT:       DEFW    VARIABLE
                DEFW    CDICT
; *
; *
                ; Beginn der DECLARE - Liste
; *
SYSDCL:         DEFW    VARIABLE
                DEFW    DECLST
; *
; *
CFLAG:          ; Compiler-Flag
; *
                DEFW    VARIABLE
                DEFW    0000
; *
; *
END:
; *
                DEFW    CODE
                DEFW    PUSH,CEND,CMPLW
                DEFW    MIN1,CFLAG,POKEW
                DEFW    CEND
; *
; *
DECLARE:        ; der Compiler für Deklarationen
; *
; *             Entspricht im wesentlichen dem Algorithmus von
; *             PROGRAM. Allerdings wird lediglich DECLARE -
; *             Liste nach erlaubten Schlüsselworten
; *             durchsucht.
```

```
; *
                DEFW    CODE
                DEFW    ZERO,CFLAG,POKEW
DECL1:          DEFW    GETWORD
                DEFW    SYSDCL,PEEKW,LOOKUP
                DEFW    IFZ,DECL2-1
                DEFW    EXEC
                DEFW    JUMP,DECL3-1
DECL2:          DEFW    SCRON,PRINTS
                DEFW    PUSH,COMPMSSG
                DEFW    ERR1
DECL3:          DEFW    CFLAG,PEEKW
                DEFW    IFNZ,DECL4-1
                DEFW    JUMP,DECL1-1
DECL4:          DEFW    CEND
; *
; *
STRING:
; *
; *     SYNTAX:         STRING NAME 'TEXT'
; *
; *             Wie     bei     allen   nachfolgend     beschriebenen
; *             Compilerroutinen,       die     im      DECLARE-Teil    eines
; *             Programmes      aufgerufen      werden      können,     wird
; *             zunächst    der     eingeführte     Name mit ENTER in die
; *             Namensliste     eingetragen und anschließend der der
; *             Routine     entsprechende       Speicherplatz       ab der
; *             Adresse,    die     zur     Zeit    in      SYSFIELD    steht,
; *             reserviert.     Dies    ist     bei     STRING      der     für die
; *             Speicherung     des     zugehörigen     Strings notwendige
; *             Platz.      Dem     Speicherplatz,      der     bekanntlich dem
; *             Parameterfeld       eines       Kernprogrammes      entspricht,
; *             wird        noch        das         jeweilige       Kodefeld
; *             vorangestellt.
; *
                DEFW    CODE
                DEFW    GETWORD,ENTER
                DEFW    VARADR,CMPLW
                DEFW    SYSFIELD,PEEKW
                DEFW    INC,INC,CMPLW
                DEFW    SCANWR
                DEFW    IFZ,COMP41-1
                DEFW    MVTEXT
                DEFW    CEND
; *
; *
CONST:
; *
; *     SYNTAX:         CONST NAME WERT
; *
; *             Bei     dieser      Routine     wird     das     Kodefeld     mit     CNSTNT
; *             gefüllt,    die     zugehörige      Konstante       zu      einer     16
; *             Bit-Zahl    konvertiert     und     in      das     Parameterfeld
; *             eingetragen.
; *
```

```
                DEFW    CODE
                DEFW    GETWORD,ENTER   ; NAME
                DEFW    CNSTADR,CMPLW
                DEFW    GETWORD,ASCBIN  ; WERT
                DEFW    IFZ,COMP41-1
                DEFW    CMPLW
                DEFW    CEND
; *
; *
CNSTADR:        DEFW    CNSTNT
                DEFW    CNSTNT
; *
; *
VAR:
; *
; *     SYNTAX:         VAR NAME
; *
; *               Hier    lautet    das    Kodefeld    VARIABLE.    Das
; *               Parameterfeld wird mit einer Null gefüllt.
; *
                DEFW    CODE
                DEFW    GETWORD,ENTER
                DEFW    VARADR,CMPLW
                DEFW    ZERO,CMPLW
                DEFW    CEND
; *
; *
VARADR:         DEFW    CNSTNT
                DEFW    VARIABLE
; *
; *
ARRAY:
; *
; *     SYNTAX:         ARRAY NAME WERT
; *
; *               Auch  hier  lautet  das  Kodefeld  VARIABLE.  Die
; *               Routine    ARR    reserviert    den    angegebenen
; *               Speicherplatz.
; *
                DEFW    CODE
                DEFW    GETWORD,ENTER
                DEFW    VARADR,CMPLW
                DEFW    GETWORD,ASCBIN
                DEFW    IFZ,COMP41-1
                DEFW    ARR
                DEFW    CEND
; *
; *
ARR:            ; reserviert den Speicher
; *
                DEFW    $+2
                POP     BC
                LD      HL,(SYSFIELD+2)
                ADD     HL,BC
                ADD     HL,BC
```

```
                LD      (SYSFIELD+2),HL
                JP      NEXT
; *
; *
VARF:
; *
; *             Auch   hier   wird   in   das  Kodefeld  VARIABLE
; *             eingetragen.  Von  MFCB2  wird  das  Muster eines
; *             DMA-Puffers  samt FCB nach (SYSFIELD) kopiert und
; *             anschließend   der   gewünschte   Namen   dort
; *             eingetragen.
; *
                DEFW    CODE
                DEFW    GETWORD,DUP
                DEFW    ENTER,VARADR,CMPLW
                DEFW    MFCB2
                DEFW    CEND
; *
MFCB2:
; *
                DEFW    $+2
                LD      DE,FCB2
                LD      HL,(SYSFIELD+2)
                EX      DE,HL
                LD      BC,00A5H
                LDIR
                POP     DE
                LD      HL,(SYSFIELD+2)
                LD      BC,0081H
                ADD     HL,BC
                EX      DE,HL
                LD      A,(HL)
                INC     HL
                CP      09H
                JR      C,LNG8
                LD      A,08H
LNG8:           LD      B,00
                LD      C,A
                LD      A,(HL)
                SUB     40H
                LD      (DE),A
                INC     DE
                LDIR
                LD      DE,00A5H
                LD      HL,(SYSFIELD+2)
                ADD     HL,DE
                LD      (SYSFIELD+2),HL
                JP      NEXT
; *
; *
ERR1:           ; Fehlerroutine 1
; *
                DEFW    CODE
                DEFW    SYSNAMES,PEEKW,DUP
                DEFW    PEEKB,ADD
```

```
                DEFW    INC,DUP,INC
                DEFW    INC
                DEFW    SYSNAMES,POKEW
                DEFW    PEEKW
                DEFW    SYSFIELD,POKEW
                DEFW    ERR,CEND
; *
; *
ERR:            ; Fehlerroutine 2
; *
                DEFW    CODE
                DEFW    PRINTS
                DEFW    MIN1,CFLAG,POKEW
                DEFW    MIN1,BUFF,POKEW
                DEFW    NEW
; *
; *
ENTER:          ; trägt einen Namen in die
                ; Namensliste ein
; *
; *
                DEFW    CODE
                DEFW    SYSFIELD,PEEKW
                DEFW    SYSNAMES,PEEKW
                DEFW    DEC,DEC,POKEW
                DEFW    DUP,PEEKB
                DEFW    DUP,INC
               DEFW    INC,INC
                DEFW    SYSNAMES,PEEKW
                DEFW    SWAP,SUB
                DEFW    DUP,SYSFIELD,PEEKW
                DEFW    SWAP,IFGT,ENT1-1
                DEFW    DUP,SYSNAMES,POKEW
                DEFW    SWAP,INC,MVBYTES
                DEFW    CEND
ENT1:           DEFW    PUSH,ERRMSG
                DEFW    ERR
                DEFW    CEND
; *
ERRMSG:         DEFB    11H
                DEFB    'Dictionary full !'
; *
; *
CMPLW:          ; trägt ein Wort in den
                ; Kodebereich ein
; *
; *
                DEFW    CODE
                DEFW    SYSFIELD,PEEKW
                DEFW    DUP,INC,INC
                DEFW    DUP,SYSNAMES,PEEKW
                DEFW    IFGT,CMP1-1
                DEFW    SYSFIELD,POKEW
                DEFW    POKEW
                DEFW    CEND
```

```
CMP1:           DEFW    PUSH,ERRMSG
                DEFW    ERR1
                DEFW    CEND
; *
; *
LIST:
; *
                DEFW    CODE
                DEFW    SYSNAMES,PEEKW
LIST1:          DEFW    DUP,PUSH,NAMES
                DEFW    IFEQ,LIST2-1
                DEFW    DRNAM,SPADR,BINASC
                DEFW    PRINTS,LAD04,SPADR
                DEFW    PRINT,CR
                DEFW    JUMP,LIST1-1
LIST2:          DEFW    POP
                DEFW    CEND
; *
; *
; *
DRNAM:          ; gibt die Namen aus
; *
                DEFW    $+2
                POP     HL
                LD      B,(HL)
                PUSH    BC
DRNAM1:         INC     HL
                LD      C,(HL)
                CALL    OUTPUT
                DJNZ    DRNAM1
                POP     BC
                INC     HL
                LD      E,(HL)
                INC     HL
                LD      D,(HL)
                INC     HL
                PUSH    HL
                LD      H,D
                LD      L,E
                INC     HL
                INC     HL
                PUSH    HL
                PUSH    DE
                LD      A,B
                NEG
                ADD     A,1AH
                LD      B,A
                JP      NEXT
; *
; *
SPADR:          ; gibt Spaces aus
; *
                DEFW    $+2
SPADR1:         LD      C,20H
                CALL    OUTPUT
```

```
                DJNZ    SPADR1
                JP      NEXT
; *
; *
LAD04:
; *
                DEFW    $+2
                LD      B,04
                JP      NEXT
; *
; *
MEMORY:
; *
                DEFW    CODE
                DEFW    PUSH,MEMMES,PRINTS
                DEFW    SYSNAMES,PEEKW
                DEFW    SYSFIELD,PEEKW
                DEFW    SUB,BINASC,PRINTS
                DEFW    CEND
; *
; *
; *
MEMMES:         DEFB    0FH
                DEFB    0DH,0AH,'Free Space : '
; *
; *
SAVE:
; *
; *     SYNTAX:         SAVE NAME
; *     BESCHREIBUNG:   SPEICHERT AKTUELLE DATEI AB
; *
                DEFW    CODE
                DEFW    XSAVE1
                DEFW    GETWORD
                DEFW    XSAVE2
                DEFW    CEND
; *
; *
XSAVE1:
; *
                DEFW    $+2
                LD      HL,EXIT
                LD      (INTLOOP+6),HL
                LD      HL,INTLOOP-1
                LD      (STRTCP+1),HL
                LD      HL,0000
                LD      (NSTART-3),HL
                LD      (NSTART-2),HL
                LD      (INTLOOP+8),HL
                LD      HL,(SYSNAMES+2)
                LD      E,(HL)
                LD      D,00
                ADD     HL,DE
                INC     HL
                LD      E,(HL)
```

```
                INC     HL
                LD      D,(HL)  ; DE:=STARTADRESSE
                EX      DE,HL
                LD      (EX+2),HL
                LD      A,(ACTDRV)
                LD      (FCB),A
                JP      NEXT
; *
; *
XSAVE2:
; *
                DEFW    $+2
        ; FCB(1...8)=BLANK
                LD      A,20H
                LD      B,8
                LD      HL,FCB
XSV21:          INC     HL
                LD      (HL),A
                DJNZ    XSV21
        ; FCB(9...11)=COM
                INC     HL
                LD      (HL),'C'
                INC     HL
                LD      (HL),'O'
                INC     HL
                LD      (HL),'M'
        ; FCB(1...7!)=NAME
                POP     HL      ; LÄNGE UND ORT DES NAMENS
                LD      A,(HL)
                AND     07
                LD      B,A
                LD      DE,FCB
XSV22:          INC     HL
                INC     DE
                LD      A,(HL)
                LD      (DE),A
                DJNZ    XSV22
        ; SET DMA ADRESS
                LD      C,1AH
                LD      DE,0080H
                CALL    BTRBSYS
        ; DELETE FILE
                LD      HL,FCB+0CH
                XOR     A
                LD      B,04
XSV23:          LD      (HL),A
                INC     HL
                DJNZ    XSV23
                ; CR AUCH NULL
                LD      DE,0010H
                ADD     HL,DE
                LD      (HL),A
                LD      DE,FCB
                LD      C,13H
                CALL    BTRBSYS
```

```
        ; MAKE FILE
                LD      C,16H
                CALL    BTRBSYS
                CP      0FFH
                JR      NZ,XSV24
                LD      C,09H
                LD      DE,DISKFULL
                CALL    BTRBSYS
                JP      NEXT
XSV24:          LD      HL,(SYSFIELD+2)
                LD      DE,0100H
                AND     A
                SBC     HL,DE
                ADD     HL,HL
                LD      A,L
                OR      A
                JR      Z,XSV25
                INC     H
XSV25:          LD      C,H     ; ANZAHL DER BLÖCKE
                LD      HL,0100H
                INC     C
XSV26:          DEC     C
                JR      Z,XSV28 ; BEI NULL --> FERTIG
                LD      DE,0080H
                LD      B,80H
XSV27:          LD      A,(HL)
                LD      (DE),A
                INC     HL
                INC     DE
                DJNZ    XSV27
        ; WRITE FILE
                LD      DE,FCB
                LD      B,C
                LD      C,15H
                CALL    BTRBSYS
                LD      C,B
                JR      XSV26
        ; CLOSE FILE
XSV28:          LD      C,10H
                LD      DE,FCB
                CALL    BTRBSYS
        ; RESTAURATION DER INT-SCHLEIFE
                LD      HL,NEWSTRT-1
                LD      (STRTCP+1),HL
                LD      HL,JUMP
                LD      (INTLOOP+6),HL
                LD      HL,INTLOOP-1
                LD      (INTLOOP+8),HL
                LD      HL,INTERACT
                LD      (EX+2),HL
                JP      NEXT
; *
RAMEND          EQU     $
; *
; *
```

```
                .DEPHASE
; *
; *
MVLSTKRN:
                LD      HL,FIELD                ; QUELLE
                LD      DE,CMPEND               ; ZIEL
                LD      BC,RAMEND-NAMES         ; ANZAHL
                LDIR
                RET
; *
; *
; *
                END     START
; *
; *
; *
```

5.2 Das Hexdump

```
     0 1  2 3  4 5  6 7  8 9  A B  C D  E F

0100 3A04 0032 2A01 CD1B 182A 0100 1109 0019  #:..2*.M..*......
0110 225A 0322 6E03 221A 0321 7103 3600 318B  #"Z."n."..!q.6.1.
0120 0121 2A01 228B 01C3 DF01 0077 C237 012F  #.!*."..C_..wB7./
0130 02CC 0271 022C 01BB 02ED C700 0000 0000  #.L.q.,.;.mG.....
0140 0000 0000 0000 0000 0000 0000 0000 0000  #................
0150 0000 0000 0000 0000 0000 0000 0000 0000  #................
0160 0000 0000 0000 0000 0000 0000 0000 0000  #................
0170 0000 0000 0000 0000 0000 0000 0000 0000  #................
0180 0000 0000 0000 0000 0000 0000 008F 0100  #................
0190 0000 0000 0000 0000 0000 0000 0000 0000  #................
01A0 0000 0000 0000 0000 0000 0000 0000 0000  #................
01B0 0000 0000 0000 0000 0000 0000 0000 0000  #................
01C0 0000 0000 0000 0000 0000 0000 0000 0000  #................
01D0 0000 0000 0000 0000 0000 0000 0000 002A  #...............*
01E0 8B01 235E 2356 228B 01EB 5E23 56EB E92A  #..#^#V"..k^#Vki*
01F0 8B01 EB22 8B01 2A8D 0173 2372 2322 8D01  #..k"..*..s#r#"..
0200 C3DF 0105 022A 8D01 2B56 2B5E 228D 01EB  #C_...*..+V+^"..k
0210 228B 01C3 DF01 1802 E123 E518 C21F 02E1  #"..C_...a#e.B..a
0220 2BE5 C3DF 0127 02E1 5E16 00D5 C3DF 0131  #+eC_.'.a^..UC_.1
0230 02E1 5E23 56D5 C3DF 013B 02E1 D173 C3DF  #.a^#VUC_.;.aQsC_
0240 0143 02E1 D173 2372 C3DF 014D 022A 8B01  #.C.aQs#rC_.M.*..
0250 235E 2356 D522 8B01 C3DF 015D 02E1 C3DF  ##^#VU"..C_.ü.aC_
0260 0163 02E1 E3E5 C3DF 016B 02E1 E5E5 C3DF  #.c.aceC_.k.aeeC_
0270 0173 022A 8B01 235E 2356 EB22 8B01 C3DF  #.s.*..#^#Vk"..C_
0280 0183 02E1 7CB5 28EB 2A8B 0123 2322 8B01  #...aö5(k*..##"..
0290 C3DF 0195 02E1 7CB5 20D9 18EC 9E02 E1D1  #C_...aö5 Y.l..aQ
02A0 A7ED 5228 CE18 E1A9 02E1 D1A7 ED52 38C3  #'mR(N.a).aQ'mR8C
02B0 18D6 EB23 5E23 56D5 C3DF 0113 D5C3 DF01  #.Vk#^#VUC_..UC_.
02C0 B202 0000 B202 0100 B202 FFFF CE02 E15E  #2...2...2...N.a^
02D0 2356 EBE9 D602 E1D1 D5E5 1ABE 2010 4723  ##VkiV.aQUe.> .G#
02E0 131A BE20 0910 F821 FFFF E5C3 DF01 2100  #..> ..x!..eC_.!.
02F0 00E5 C3DF 01F7 02C1 CD35 03C3 DF01 0003  #.eC_.w.AM5.C_...
0300 CD72 036F 2600 E5C3 DF01 0C03 2118 0322  ##Mr.o&.eC_...!.."
0310 5A03 226E 03C3 DF01 C5CD 0000 C159 0E05  #Z."n.C_.EM..AY..
0320 C391 0325 032A 0100 1109 0019 225A 0322  #C..%.*......."Z."
0330 6E03 C3DF 01C5 D5E5 CD43 03E1 D1C1 C979  #n.C_.EUeMC.aQAIy
0340 C34A 0821 7103 79FE 0920 150E 207E 46C6  #CJ.!q.yß. .. ßFF
0350 08E6 F877 B8C8 04C5 F5CD 0000 F1C1 18F4  #.fxwßH.EuM..qA.t
0360 FE0D 2004 3600 1805 FE0A 28F8 34CD 0000  #ß. .6...ß.(x4M..
0370 C900 0E01 C391 03C3 4A08 C97D 0321 7A03  #I...C..CJ.Iü.!z.
0380 2239 03C3 DF01 8803 2143 0322 3903 C3DF  #"9.C_...!C."9.C_
0390 01E5 D5C5 CD05 00C1 D1E1 C99D 032A 8B01  #.eUEM..AQaI..*..
03A0 23E5 4E06 0009 228B 01E1 4123 4ECD 3503  ##eN..."..aA#NM5.
03B0 10F9 C3DF 01B7 03D1 D51A 4721 0000 131A  #.yC_.7.QU.G!....
03C0 FE30 381D FE3A 3019 D630 D529 E529 29D1  #ß08.ß:0.V0U)e))Q
03D0 195F 1600 19D1 10E6 D1E5 21FF FFE5 C3DF  #._...Q.fQe!..eC_
03E0 0121 0000 E5C3 DF01 EA03 214D 0406 0636  #.!..eC_.j.!M...6
03F0 002B 10FB E1AF 5706 1029 8F27 5F7A 8F27  #.+.äa/W..).'_z.'
0400 57CB 117B 10F3 214D 0473 2B72 2B71 1148  #WK.ä.s!M.s+r+q.H
0410 0497 1213 ED67 1213 23ED 6F12 13ED 6F12  #....mg..#mo..mo.
0420 1323 ED6F 12ED 6F77 2148 0406 057E C620  #.#mo.mow!H...ßF
```

```
     0 1  2 3  4 5  6 7  8 9  A B  C D  E F

0430 FE20 2004 7723 10F5 047E F630 7723 10F9 #ß   .w#.u.ßv0w#.y
0440 2147 04E5 C3DF 0106 0000 0000 0000 5004 #!G.eC_........P.
0450 E1D1 19E5 C3DF 0159 04D1 E1A7 ED52 E5C3 #aQ.eC_.Y.Qa'mReC
0460 DF01 EF01 7404 5B02 0302 EF01 7404 6102 #_.o.t.Ä...o.t.a.
0470 5B02 0302 7604 C1D1 2100 000B 782F 4779 #Ä...v.AQ!...x/Gy
0480 2F4F 3E10 29F5 EB29 EB30 012C F138 0CE5 #/O>.)uk)k0.,q8.e
0490 0938 03E1 1807 1C33 3318 0209 1C3D 20E4 #.8.a...33....= d
04A0 D5E5 C3DF 01A7 04E1 D106 104A 7BEB 2100 #UeC_.'.aQ..Jäk!.
04B0 00CB 391F 3001 19EB 29EB 10F5 E5C3 DF01 #.K9.0..k)k.ueC_.
04C0 C204 C1D1 E1ED B0C3 DF01 EF01 1805 2F02 #B.AQam0C_.o.../.
04D0 4B02 1F00 A504 4B02 4E07 6A04 4B02 0100 #K...%.K.N.j.K...
04E0 4E04 1805 4102 1C05 2F02 4B02 1D00 A504 #N...A.../.K...%.
04F0 4B02 C106 6A04 4B02 0100 4E04 1C05 4102 #K.A.j.K...N...A.
0500 1C05 2F02 4B02 0500 6204 1805 2F02 4B02 #../.K...b.../.K.
0510 0D00 6204 A504 0302 BB02 7B00 BB02 EA00 #..b.%...;.ä.;.j.
0520 EF01 6102 1602 7005 1D02 7005 0302 3005 #o.a...p...p...0.
0530 CD79 0513 D5CD 7905 E1E5 A7ED 5228 0ECD #My..UMy.ae'mR(.M
0540 6505 D1CD 6505 2100 00E5 C3DF 01E1 21FF #e.QMe.!..eC_.a!.
0550 FFE5 C3DF 0157 0511 FFFF 2A84 0519 562B #.eC_.W....*...V+
0560 5ED5 C3DF 012A 8405 7323 7223 2284 05C9 #^UC_.*..s#r#"..I
0570 7205 D1CD 6505 C3DF 012A 8405 2B56 2B5E #r.QMe.C_.*..+V+^
0580 2284 05C9 8605 0000 0000 0000 0000 0000 #"..I............
0590 0000 0000 0000 0000 0000 0000 0000 0000 #................
05A0 0000 0000 0000 A805 0E0D CD35 030E 0ACD #......(...M5...M
05B0 3503 C3DF 01EF 012F 02E8 03BF 0503 02C1 #5.C_.o./.h.?...A
05C0 05E1 4623 4ECD 3503 10F9 C3DF 01EF 012F #.aF#NM5..yC_.o./
05D0 02BF 0503 02D7 053E 2032 1306 ED5B EE06 #.?...W.> 2..mÄn.
05E0 9712 1817 E605 3E00 1804 EC05 3E20 3213 #....f.>...l.> 2.
05F0 06ED 5BEE 06E1 9712 7323 72D5 E113 CD72 #.mÄn.a..s#rUa.Mr
0600 03FE 0820 091B 7EB7 28F3 1B35 18EF FE60 #.ß. ..ß7(s.5.oß`
0610 3802 D620 ED53 EE06 FE0D CADF 0134 1218 #8.V mSn.ß.J_.4..
0620 DC23 060E 061E FFCD 9103 B728 FA5F 1600 #Ö#.....M..7(z_..
0630 ED53 3906 C3DF 01BB 0200 003D 06E1 D1A7 #mS9.C_.;...=.aQ'
0640 ED52 2100 0020 012B E5C3 DF01 4E06 E17C #mR!.. .+eC_.N.aö
0650 B518 EF55 06E1 D1C3 DA02 5C06 E1D1 461A #5.oU.aQCZ.Ö.aQF.
0660 B830 0147 2313 1A4E B928 0430 0418 0910 #80.G#..N9(.0....
0670 F321 FFFF E5C3 DF01 2100 00E5 C3DF 0181 #s!..eC_.!..eC_..
0680 06D1 E1A7 ED52 ED62 E5C3 DF01 8E06 E1D1 #.Qa'mRmbeC_...aQ
0690 18F1 9406 D1E1 A7ED 523F 18EA 9E06 E1D1 #.q..Qa'mR?.j..aQ
06A0 18F4 EF01 3B06 C206 0302 AC06 E1D1 7CA2 #.to.;.B...,.aQö"
06B0 676F E5C3 DF01 B806 E1D1 7CB2 676F E5C3 #goeC_.8.aQö2goeC
06C0 DF01 C406 E129 3FED 62E5 C3DF 01CF 062A #_.D.a)?mbeC_.O.*
06D0 CA02 E5C3 DF01 D806 2AC2 02E5 C3DF 01E1 #J.eC_.X.*B.eC_.a
06E0 0611 8000 0E1A CD91 03C3 0000 BB02 960B #......M..C..;...
06F0 F206 E111 8000 193E 0077 23E5 110C 0019 #r.a....>.w#e....
0700 AF06 0477 2310 FC11 1000 1977 D10E 0FCD #/..w#.ö....wQ..M
0710 9103 FEFF C2DF 010E 16CD 9103 FEFF C2DF #..ß.B_...M..ß.B_
0720 010E 0911 2C07 CD91 03C3 DF01 0D0A 4449 #....,.M..C_...DI
0730 534B 2046 554C 4C20 2400 0000 0000 0000 #SK FULL $.......
0740 0000 0000 0000 0000 0000 0000 0000 0000 #................
0750 0000 0000 0000 0000 0000 0000 0000 0000 #................
0760 0000 0000 0000 0000 0000 0000 0000 0000 #................
0770 0000 0000 0000 0000 0000 0000 0000 0000 #................
```

```
     0 1  2 3  4 5  6 7  8 9  A B  C D  E F

0780 0000 0000 0000 0000 0000 0000 0000 0000 #................
0790 0000 0000 0000 0000 0000 0000 0000 0000 #................
07A0 0000 0000 0000 0000 0000 0000 0000 0000 #................
07B0 0000 0000 0000 0000 0000 0020 2020 2020 #............
07C0 2020 2052 504E 0000 0000 0020 2020 2020 #    RPN.....
07D0 2020 2020 2020 0000 0000 0000 0000 E007 #        ........`.
07E0 0E10 E111 8000 193E 0077 23EB CD91 03C3 #..a....>.w#kM..C
07F0 DF01 F407 E111 8100 19EB 0E13 CD91 03C3 #_.t.a....k..M..C
0800 DF01 0408 D1D5 0E1A CD91 030E 14D1 2181 #_...QU..M....Q!.
0810 0019 EBCD 9103 B728 0721 0000 E5C3 DF01 #..kM..7(.!..eC_.
0820 21FF FFE5 C3DF 0129 08E1 2285 0811 8000 #!..eC_.).a".....
0830 1922 8708 3E00 323E 033E CD32 7403 214A #."..>.2>.>M2t.!J
0840 0822 4103 2278 03C3 DF01 F5C5 D5E5 2A87 #."A."x.C_.uEUe*.
0850 085E 1600 2A85 0819 777B 3CFE 8023 361A #.^..*...wä<ß.#6.
0860 201A ED5B 8508 0E1A CD91 03ED 5B87 0813 # .mÄ....M..mÄ...
0870 0E15 CD91 032A 8508 361A 3E00 2A87 0877 #..M..*..6.>.*..w
0880 E1D1 C1F1 C900 0000 00CB 08E1 2244 0911 #aQAqI......a"D..
0890 8000 1922 4609 ED5B EE06 E197 1273 2372 #..."F.mÄn.a..s#r
08A0 D5E1 13ED 53EE 0622 4809 2A46 095E 1600 #Ua.mSn."H.*F.^..
08B0 2A44 0919 7B1C 3CFE 807E 01FF FF20 0501 #*D..ä.<ß.ß... ..
08C0 0000 1E00 2A46 0973 FE1A 200D 2A46 0936 #....*F.sß. .*F.6
08D0 0001 0000 C5C5 C3DF 01C5 4FCD 3503 79C1 #....EEC_.EOM5.yA
08E0 FE0A 2846 FE0D 2039 2A48 0997 BE20 0D34 #ß.(Fß. 9*H..> .4
08F0 3E20 ED5B EE06 1213 ED53 EE06 2A46 0934 #> mÄn...mSn.*F.4
0900 7EFE 0128 06FE 8020 0B36 0001 FFFF C501 #ßß.(.ß. .6....E.
0910 0000 1804 01FF FFC5 C50E 0ACD 3503 C3DF #.......EE..M5.C_
0920 012A 4809 ED5B EE06 3412 C578 B7C2 4009 #.*H.mÄn.4.Ex7B§.
0930 01FF FFC5 ED5B EE06 13ED 53EE 06C3 DF01 #...EmÄn..mSn.C_.
0940 C1C3 A208 0000 0000 0000 4C09 3EC9 323E #AC".......L.>I2>
0950 033E C332 7403 C3DF 015B 09D1 D50E 1ACD #.>C2t.C_.Ä.QU..M
0960 9103 0E15 D121 8100 19EB CD91 03C3 DF01 #....Q!...kM..C_.
0970 7209 E106 807E 23FE 1ACA DF01 4FCD 3503 #r.a..ß#ß.J_.OM5.
0980 10F3 C3DF 0187 0911 8000 0E1A CD91 033E #.sC_........M..>
0990 0032 E509 112D 0A0E 09CD 9103 3A2A 013C #.2e..-...M..:*.<
09A0 321D 0AC6 404F CD35 030E 3ACD 3503 11E6 #2..F§OM5..:M5..f
09B0 090E 09CD 9103 111D 0A0E 11CD 9103 CDF0 #...M.......M..Mp
09C0 093A E509 3CFE 0420 0A11 E609 0E09 CD91 #.:e.<ß. ..f...M.
09D0 033E 0032 E509 111D 0A0E 12CD 9103 FEFF #.>.2e......M..ß.
09E0 CADF 0118 D900 0D0A 2020 2020 2020 2024 #J_..Y...        $
09F0 6F26 0029 2929 2929 1180 0019 0608 234E #o&.)))))......#N
0A00 CD35 0310 F90E 2ECD 3503 0603 234E CD35 #M5..y..M5...#NM5
0A10 0310 F906 060E 20CD 3503 10FB C900 3F3F #..y... M5..äI.??
0A20 3F3F 3F3F 3F3F 3F3F 3F00 0000 0020 2020 #??????????....
0A30 2020 4461 7465 6965 6E20 6175 6620 6465 #  Dateien auf de
0A40 6D20 4C61 7566 7765 726B 2024 4E0A 117F #m Laufwerk $N...
0A50 0A0E 09CD 9103 3A2A 01C6 414F CD35 0311 #...M..:*.FAOM5..
0A60 9D0A 0E09 CD91 03CD 7203 FE60 3802 D620 #....M..Mr.ß`8.V
0A70 D641 FA5F 0AFE 0830 E632 2A01 C3DF 010D #VAz_.ß.0f2*.C_..
0A80 0A20 2020 2020 416B 7475 656C 6C65 7320 #.      Aktuelles
0A90 2020 4C61 7566 7765 726B 3A20 240D 0A20 #  Laufwerk: $..
0AA0 2020 2020 4765 777D 6E73 6368 7465 7320 #    Gewünschtes
0AB0 4C61 7566 7765 726B 3A20 24BD 0AE1 E511 #Laufwerk: $=.ae.
0AC0 8200 1906 09CD 6F0B FE1B 200F E111 8200 #.....Mo.ß. .a...
```

```
      0 1  2 3  4 5  6 7  8 9  A B  C D  E F

0AD0 1906 0B36 2023 10FB C3DF 01FE 2E28 2DFE #...6 #.äC_.ß.(-ß
0AE0 6038 02D6 20FE 0820 1078 FE09 28D7 0E08 #`8.V ß. .xß.(W..
0AF0 CD35 0304 2B36 2018 CC77 234F CD35 0310 #M5..+6 .Lw#OM5..
0B00 C411 4B0B 0E09 CD91 03E1 18B2 4FCD 3503 #D.K...M..a.2OM5.
0B10 0528 063E 2077 2310 F806 03CD 6F0B FE1B #.(.> w#.x..Mo.ß.
0B20 2003 E118 99FE 6038 02D6 20FE 0820 1078 # .a..ß`8.V ß. .x
0B30 FE03 28E7 0E08 CD35 0304 2B36 2018 DC77 #ß.(g..M5..+6 .Čw
0B40 234F CD35 0310 D4E1 C3DF 010D 2020 2020 ##OM5..TaC_..
0B50 2055 6E67 7D6C 7469 6765 7220 4E61 6D65 # Ungültiger Name
0B60 2020 2020 2020 2020 2020 2020 2020 241E #                $.
0B70 FF0E 06CD 9103 FE21 3801 C9FE 08C8 FE2E #...M..ß!8.Iß.Hß.
0B80 C8FE 1BC8 18E9 880B 21F0 0622 EE06 3E00 #Hß.H.i..!p."n.>.
0B90 327C C1C3 DF01 0443 4F44 45CB C304 4341 #2|ÃAC_..CODEKC.CA
0BA0 4C4C C3C3 013F 2F02 043F 2842 2925 0202 #LLCC.?/..?(B)%..
0BB0 3A3D 4102 053A 3D28 4229 3902 0349 4E43 #:=A..:=(B)9..INC
0BC0 1602 0344 5550 6902 0344 4543 1D02 012B #...DUPi..DEC...+
0BD0 4E04 012D 5704 0344 4956 6204 012A A504 #N..-W..DIVb..*%.
0BE0 034D 4F44 6A04 0243 52A6 0503 4E4F 54C2 #.MODj..CR&..NOTB
0BF0 0603 414E 44AA 0602 4F52 B606 0750 524F #..AND*..OR6..PRO
0C00 4752 414D D7C8 064F 5554 5055 5435 0301 #GRAMWH.OUTPUT5..
0C10 3D3B 0602 303D 4C06 013C 7F06 013E 8C06 #=;..0=L..<...>..
0C20 023E 3D92 0602 3C3D 9C06 023C 3EA2 0601 #.>=...<=...<>"..
0C30 4955 0504 5357 4150 6102 043D 2853 2953 #IU..SWAPa..=(S)S
0C40 0605 3E3D 2853 295A 0608 5052 494E 5428 #..>=(S)Z..PRINT(
0C50 5329 CD05 0752 4541 4428 5329 E405 0852 #S)M..READ(S)d..R
0C60 4541 4428 4348 29EA 0506 494E 4348 4152 #EAD(CH)j..INCHAR
0C70 2106 074F 5554 4348 4152 F502 0550 5249 #!..OUTCHARu..PRI
0C80 4E54 B505 034E 4557 FFC3 0454 5255 45CD #NT5..NEW.C.TRUEM
0C90 0605 4641 4C53 45D6 0607 4445 434C 4152 #..FALSEV..DECLAR
0CA0 4582 C907 434F 4D50 494C 4520 C406 5343 #E.I.COMPILE D.SC
0CB0 524F 4646 7B03 0553 4352 4F4E 8603 0653 #ROFFä..SCRON...S
0CC0 5953 5445 4DDF 0608 5359 5346 4945 4C44 #YSTEM_..SYSFIELD
0CD0 EC06 0742 5452 4253 5953 9103 075A 4549 #l..BTRBSYS...ZEI
0CE0 4348 454E 3706 0444 524F 4E0A 0305 4452 #CHEN7..DRON...DR
0CF0 4F46 4623 0304 4C49 5354 1CCB 064D 454D #OFF#..LIST.K.MEM
0D00 4F52 5979 CB03 524E 44CA 0404 5341 5645 #ORYyK.RNDJ..SAVE
0D10 A1CB 0546 494C 4553 8509 0843 4847 4452 #!K.FILES...CHGDR
0D20 4956 454C 0A09 4441 5445 494E 414D 45BB #IVEL..DATEINAME;
0D30 0A05 5245 4144 4602 0804 4F50 454E F006 #..READF...OPENp.
0D40 0543 4C4F 5345 DE07 0644 454C 4554 45F2 #.CLOSE^..DELETEr
0D50 0706 5752 4954 4546 5909 0650 5249 4E54 #..WRITEFY..PRINT
0D60 4670 0907 5457 5249 5445 4627 0806 5452 #Fp..TWRITEF´..TR
0D70 4541 4446 8908 0454 454E 444A 0906 4143 #EADF...TENDJ..AC
0D80 5444 5256 2A01 074E 4F46 494C 4553 860B #TDRV*..NOFILES..
0D90 0004 4345 4E44 B9C3 0249 46CC C604 5448 #..CEND9C.IFLF.TH
0DA0 454E DEC6 0445 4C53 45EC C606 5245 5045 #EN^F.ELSEIF.REPE
0DB0 4154 0AC7 0555 4E54 494C 14C7 044C 4F4F #AT.G.UNTIL.G.LOO
0DC0 5026 C703 464F 523E C703 454E 4472 C905 #P&G.FOR>G.ENDrI.
0DD0 5752 4954 45A6 C604 5245 4144 05C5 0003 #WRITE&F.READ.E..
0DE0 5641 52EC C904 5641 5246 24CA 0543 4F4E #VARlI.VARF$J.CON
0DF0 5354 D2C9 0541 5252 4159 00CA 0653 5452 #STRI.ARRAY.J.STR
0E00 494E 47B4 C904 4445 4E44 72C9 00EF 014B #ING4I.DENDrI.o.K
0E10 0281 C2BF 0503 0293 0D0A 3E3E 2020 2020 #..B?......>>
```

```
      0 1  2 3  4 5  6 7  8 9  A B  C D  E F

0E20 5250 4E4C 202D 2043 4F4D 5049 4C45 5220  #RPNL - COMPILER
0E30 2020 2020 2020 2020 2020 5645 5253 2E20  #           VERS.
0E40 312E 3020 2020 203C 3C0D 0A3E 3E20 2020  #1.0     <<..>>
0E50 2043 6F70 7972 6967 6874 2020 2020 4469  # Copyright    Di
0E60 706C 2E2D 496E 672E 2047 2E20 576F 7374  #pl.-Ing. G. Wost
0E70 7261 636B 2020 2020 3C3C 0D0A 3E3E 2020  #rack     <<..>>
0E80 2020 2020 2020 2020 2020 2020 2020 2020  #
0E90 2020 2020 2020 2020 2020 2020 2020 2020  #
0EA0 2020 2020 2020 2020 203C 3CEF 01C0 026E  #          <<o.$.n
0EB0 C941 0245 C59E CA03 02EF 0145 C575 C381  #IA.EE.J..o.EEuC.
0EC0 023C C3F6 CAEC 062F 021D 02EC 0641 0271  #.<CvJl./...l.A.q
0ED0 0266 C362 C92F 02AF C781 024C C3F6 CA71  #.fCbI/./G..LCvJq
0EE0 0266 C366 C92F 02AF C781 025C C3CC 0271  #.fCfI/./G..ÖCL.q
0EF0 0266 C386 03BF 054B 0237 C96A CA6E C92F  #.fC..?.K.7IjJnI/
0F00 0293 0272 C371 0224 C303 0277 C3D1 D51A  #...rCq.$C..wCQU.
0F10 0602 B820 31AF 21B7 C377 2377 131A 2BD6  #..8 1/!7Cw#w..+V
0F20 30FA B0C3 FE0A 380C D607 FE0A FAB0 C3FE  #0z0CØ.8.V.Ø.z0CØ
0F30 10F2 B0C3 13ED 6F23 ED6F 10E1 E12A B7C3  #.r0C.mo#mo.aa*7C
0F40 E52A CA02 1803 2AC2 02E5 C3DF 0100 00EF  #e*J...*B.eC_...o
0F50 01C8 026E C941 0203 02EF 0115 C323 C303  #.H.nIA...o..C#C.
0F60 02EF 0115 C3EC 062F 0269 0216 0216 0269  #.o..Cl./.i.....i
0F70 02EC 0641 0261 0241 0223 C3E9 C303 02EB  #.l.A.a.A.#CiC..k
0F80 C32A EE06 3EC3 7711 DF01 2373 2372 2322  #C*n.>Cw._.#s#r#"
0F90 EE06 C3DF 0101 C43E C932 3E03 3EC3 3274  #n.C_..D>I2>.>C2t
0FA0 0321 9103 2275 0321 00C0 2264 C921 960B  #.!.."u.!.$"dI!..
0FB0 22EE 06C3 0901 EF01 45C5 2AC4 58C4 0302  #"n.C..o.EE*DXD..
0FC0 2CC4 1180 000E 1ACD 9103 21E1 C411 5C00  #,D.....M..!aD.Ö.
0FD0 0124 00ED B0E1 115C 007E FE09 3802 3E08  #.$.mØa.Ö.ØØ.8.>.
0FE0 0600 4F23 7ED6 4012 13ED B0C3 DF01 5AC4  #..O#ØV$..mØC_.ZD
0FF0 115C 000E 0FCD 9103 FEFF 200B 1180 C40E  #.Ö...M..Ø. ...D.
1000 09CD 9103 C3DF 0121 89C4 2275 033E 8032  #.M..C_.!.D"u.>.2
1010 7FC4 C321 0100 4E4F 2046 494C 4520 243A  #.DC!..NO FILE $:
1020 7FC4 FE80 2012 115C 000E 14CD 9103 B728  #.DØ. ..Ö...M..7(
1030 0721 9103 2275 03C9 5F16 003C 327F C421  #.!.."u.I_..<2.D!
1040 8000 194E CD35 0379 FE0E 2002 3E09 FE0A  #...NM5.yØ. .>.Ø.
1050 28CD FE1A C011 CCC4 0E09 CD91 033E 0DC3  #(MØ.$.LD..M..>.C
1060 9BC4 2A2A 2A43 4F44 4520 4745 4E45 5241  #.D***CODE GENERA
1070 5445 442A 2A2A 2400 2020 2020 2020 2020  #TED***$.
1080 5052 4700 0000 0000 2020 2020 2020 2020  #PRG.....
1090 2020 2000 0000 0000 0000 00EF 014B 02EC  #    ........o.K.l
10A0 06F6 CA4B 022F 02F6 CA4B 0269 02F6 CA4B  #.vJK./.vJK.i.vJK
10B0 02D5 05F6 CA4B 02EC 06F6 CA4B 0241 02F6  #.U.vJK.l.vJK.A.v
10C0 CA4B 02B5 03F6 CA4B 025B 02F6 CA4B 0261  #JK.5.vJK.A.vJK.a
10D0 02F6 CA4B 0241 02F6 CA03 02EF 0155 C593  #.vJK.A.vJ..o.UE.
10E0 0252 C5A2 C571 0246 C503 0257 C521 26C6  #.RE"Eq.FE..WE!&F
10F0 4E06 0034 2836 2309 4123 047E B728 2DFE  #N..4(6#.A#.Ø7(-Ø
1100 2028 F618 0323 047E 1127 C60E 0013 120C  # (v..#.Ø.~F.....
1110 0423 7EB7 2004 06FF 3E20 FE20 20EF 2126  #.#Ø7 ...> Ø  o!&
1120 C670 2371 E521 FFFF E5C3 DF01 3EFF 3226  #Fp#qe!..eC_.>.2&
1130 C621 0000 E5C3 DF01 EF01 C002 24C6 3902  #F!..eC_.o.$.$F9.
1140 24C6 1602 1602 6902 FE02 6902 4B02 0800  #$F....i.Ø.i.K...
1150 9C02 E5C5 6902 4B02 0900 9C02 FBC5 6902  #..eEi.K.....äEi.
1160 4B02 3B00 9C02 05C6 6902 4B02 0D00 9C02  #K.;....Fi.K.....
```

```
     0 1  2 3  4 5  6 7  8 9  A B  C D  E F

1170 11C6 6102 3902 1602 7102 AFC5 5B02 5B02 #.Fa.9...q./EÄ.Ä.
1180 1D02 4B02 2000 F502 4B02 0800 F502 7102 #..K. .u.K...u.q.
1190 AFC5 5B02 4B02 2000 7102 DBC5 FE02 4B02 #/EÄ.K. .q.ÄEß.K.
11A0 0D00 3B06 8102 05C6 5B02 5B02 C002 6102 #..;....FÄ.Ä.$.a.
11B0 3902 4B02 0A00 F502 0302 BB02 0000 0000 #9.K...u...;.....
11C0 0000 0000 0000 0000 0000 0000 0000 0000 #................
11D0 0000 0000 0000 0000 0000 0000 0000 0000 #................
11E0 0000 0000 0000 0000 0000 0000 0000 0000 #................
11F0 0000 0000 0000 0000 0000 0000 0000 0000 #................
1200 0000 0000 0000 0000 0000 0000 0000 0000 #................
1210 0000 0000 0000 0000 0000 0000 0000 0000 #................
1220 0000 0000 0000 0000 0000 0000 0000 0000 #................
1230 0000 0000 0000 0000 0000 0000 EF01 4B02 #............o.K.
1240 9B03 F6CA 62C7 8102 20C9 B8C6 0302 BAC6 #..vJbG.. I8F..:F
1250 E14E 0600 03ED 5BEE 06ED B0EB 22EE 06C3 #aN...mÄn.mØk"n.C
1260 DF01 EF01 4B02 8102 F6CA EC06 2F02 C002 #_.o.K...vJl./.$.
1270 F6CA 0302 EF01 EC06 2F02 1D02 6102 4102 #vJ..o.l./...a.A.
1280 0302 EF01 4B02 7102 F6CA EC06 2F02 C002 #..o.K.q.vJl./.$.
1290 F6CA 6102 EC06 2F02 1D02 6102 4102 0302 #vJa.l./...a.A...
12A0 EF01 EC06 2F02 1D02 0302 EF01 4B02 9302 #o.l./.....o.K...
12B0 F6CA EC06 2F02 C002 F6CA 0302 EF01 4B02 #vJl./.$.vJ..o.K.
12C0 7102 F6CA 6102 F6CA EC06 2F02 1D02 6102 #q.vJa.vJl./...a.
12D0 4102 0302 EF01 4B02 2005 F6CA EC06 2F02 #A...o.K. .vJl./.
12E0 1D02 4B02 2E05 F6CA 4B02 9302 F6CA EC06 #..K...vJK...vJl.
12F0 2F02 C002 F6CA 0302 64C7 2126 C64E 0600 #/.$.vJ..dG!&FN..
1300 3428 3623 0941 2304 7EB7 282D FE27 28F6 #4(6#.A#.ß7(-ß'(v
1310 1803 2304 7E11 27C6 0E00 1312 0C04 237E #..#.ß.'F......#ß
1320 B720 0406 FF3E 27FE 2720 EF21 26C6 7023 #7 ...>'ß' o!&Fp#
1330 71E5 21FF FFE5 C3DF 013E FF32 26C6 2100 #qe!..eC_.>.2&F!.
1340 00E5 C3DF 01EF 0169 0225 0281 02CC C7D4 #.eC_.o.i.%...LGT
1350 0293 02D2 C7E1 C74E 044B 0202 004E 0471 #...RGaGN.K...N.q
1360 02B0 C75B 02C0 0203 0261 025B 02E1 C74E #.ØGÄ.$...a.Ä.aGN
1370 042F 02C8 0203 02E3 C7E1 4E06 0023 E5C5 #./.H...cGaN..#eE
1380 C3DF 01EF 01C9 C835 C855 C581 0218 C862 #C_.o.IH5HUE...Hb
1390 C92F 02AF C781 0208 C8CC 0271 02F2 C7B5 #I/./G...HL.q.rG5
13A0 0393 02F2 C74B 0221 C8BF 05BF 0503 024B #...rGK.!H?.?...K
13B0 022F C8BF 0503 020D 4E6F 7420 6465 6669 #./H?....Not defi
13C0 6E65 643A 2005 0D0A 4F6B 21EF 01C0 0224 #ned: ...Ok!o.$.$
13D0 C639 0224 C616 0216 0269 02B7 C869 024B #F9.$F....i.7Hi.K
13E0 0208 009C 0278 C869 024B 0209 009C 028E #.....xHi.K......
13F0 C869 024B 023B 009C 0298 C869 024B 020D #Hi.K.;....Hi.K..
1400 009C 02A4 C861 0239 0216 0271 0242 C85B #...$Ha.9...q.BHÄ
1410 025B 021D 024B 0220 00F5 024B 0208 00F5 #.Ä...K. .u.K...u
1420 0271 0242 C85B 024B 0220 0071 026E C8FE #.q.BHÄ.K. .q.nHß
1430 024B 020D 003B 0681 0298 C85B 025B 02C0 #.K...;....HÄ.Ä.$
1440 0261 0239 024B 020A 00F5 0203 02B9 C8CD #.a.9.K...u...9HM
1450 7203 FE60 3802 D620 6F26 00E5 C3DF 01EF #r.ß`8.V o&.eC_.o
1460 014B 02D3 C8BF 0503 0203 0D0A 23EF 01C0 #.K.SH?......#o.$
1470 026E C941 0245 C59E CA4B 02EF 01F6 CA45 #.nIA.EE.JK.o.vJE
1480 C566 C92F 02AF C781 02FA C8CC 0271 0228 #EfI/./G..zHL.q.(
1490 C962 C92F 02AF C781 020A C9F6 CA71 0228 #IbI/./G...IvJq.(
14A0 C9B5 0381 021C C94B 024B 02F6 CAF6 CA71 #I5....IK.K.vJvJq
14B0 0228 C986 03BF 0586 034B 0237 C96A CA6E #.(I..?...K.7IjJn
```

```
      0 1  2 3  4 5  6 7  8 9  A B  C D  E F

14C0 C92F 0293 0234 C971 02E8 C803 022A 2069 #I/...4Iq.hH..* i
14D0 7320 756E 6465 6669 6E65 642E 0D0A 0D0A #s undefined.....
14E0 202A 2A2A 4E4F 2043 4F44 4520 4745 4E45 # ***NO CODE GENE
14F0 5241 5445 442A 2A2A BB02 00C0 BB02 FBC1 #RATED***;..§;.äA
1500 BB02 49C2 BB02 0000 EF01 4B02 0302 F6CA #;.IB;...o.K...vJ
1510 C802 6EC9 4102 0302 EF01 C002 6EC9 4102 #H.nIA...o.$.nIA.
1520 45C5 6AC9 2F02 AFC7 8102 9BC9 CC02 7102 #EEjI/./G...IL.q.
1530 A5C9 8603 BF05 4B02 37C9 6ACA 6EC9 2F02 #%I..?.K.7IjJnI/.
1540 9302 B1C9 7102 89C9 0302 EF01 45C5 9ECA #..1Iq..I..o.EE.J
1550 FCC9 F6CA EC06 2F02 1602 1602 F6CA 62C7 #öIvJl./.....vJbG
1560 8102 20C9 B8C6 0302 EF01 45C5 9ECA E8C9 #.. I8F..o.EE.JhI
1570 F6CA 45C5 B503 8102 20C9 F6CA 0302 B202 #vJEE5... IvJ..2.
1580 B202 EF01 45C5 9ECA FCC9 F6CA C002 F6CA #2.o.EE.JöIvJ§.vJ
1590 0302 B202 BB02 EF01 45C5 9ECA FCC9 F6CA #..2.;.o.EE.JöIvJ
15A0 45C5 B503 8102 20C9 16CA 0302 18CA C12A #EE5... I.J...JA*
15B0 EE06 0909 22EE 06C3 DF01 EF01 45C5 6902 #n..."n.C_.o.EEi.
15C0 9ECA FCC9 F6CA 34CA 0302 36CA 1139 072A #.JöIvJ4J..6J.9.*
15D0 EE06 EB01 A500 EDB0 D12A EE06 0181 0009 #n.k.%.m0Q*n.....
15E0 EB7E 23FE 0938 023E 0806 004F 7ED6 4012 #kß#ß.8.>...OßV§.
15F0 13ED B011 A500 2AEE 0619 22EE 06C3 DF01 #.m0.%.*n.."n.C_.
1600 EF01 62C9 2F02 6902 2502 4E04 1602 6902 #o.bI/.i.%.N...i.
1610 1602 1602 62C9 4102 2F02 EC06 4102 8CCA #....bIA./.l.A..J
1620 0302 EF01 BF05 C802 6EC9 4102 C802 24C6 #..o.?.H.nIA.H.$F
1630 4102 FFC3 EF01 EC06 2F02 62C9 2F02 1D02 #A..Co.l./.bI/...
1640 1D02 4102 6902 2502 6902 1602 1602 1602 #..A.i.%.i.......
1650 62C9 2F02 6102 5704 6902 EC06 2F02 6102 #bI/.a.W.i.l./.a.
1660 A702 DBCA 6902 62C9 4102 6102 1602 C004 #´.ÄJi.bIA.a...§.
1670 0302 4B02 E4CA 8CCA 0302 1144 6963 7469 #..K.dJ.J...Dicti
1680 6F6E 6172 7920 6675 6C6C 2021 EF01 EC06 #onary full !o.l.
1690 2F02 6902 1602 1602 6902 62C9 2F02 A702 #/.i.....i.bI/.´.
16A0 13CB EC06 4102 4102 0302 4B02 E4CA 6ACA #.Kl.A.A...K.dJjJ
16B0 0302 EF01 62C9 2F02 6902 4B02 00C0 9C02 #..o.bI/.i.K..§..
16C0 3FCB 44CB 66CB E803 BF05 72CB 66CB B505 #?KDKfKh.?.rKfK5.
16D0 A605 7102 21CB 5B02 0302 46CB E146 C523 #&.q.!KÄ...FKaFE#
16E0 4ECD 3503 10F9 C123 5E23 5623 E562 6B23 #NM5..yA#^#V#ebk#
16F0 23E5 D578 ED44 C61A 47C3 DF01 68CB 0E20 ##eUxmDF.GC_.hK.
1700 CD35 0310 F9C3 DF01 74CB 0604 C3DF 01EF #M5..yC_.tK..C_.o
1710 014B 0291 CBBF 0562 C92F 02EC 062F 0257 #.K..K?.bI/.l./.W
1720 04E8 03BF 0503 020F 0D0A 4672 6565 2053 #.h.?......Free S
1730 7061 6365 203A 20EF 01AB CB45 C5DD CB03 #pace : o.+KEEüK.
1740 02AD CB21 DF06 2233 0121 2C01 2222 0121 #.-K!_."3.!,.""..!
1750 0000 2206 0122 0701 2235 012A 64C9 5E16 #.."..".."5.*dI^.
1760 0019 235E 2356 EB22 3901 3A2A 0132 5C00 #..#^#Vk"9.:*.2ö.
1770 C3DF 01DF CB3E 2006 0821 5C00 2377 10FC #C_._K> ..!ö.#w.ö
1780 2336 4323 364F 2336 4DE1 7EE6 0747 115C ##6C#6O#6Maßf.G.ö
1790 0023 137E 1210 FA0E 1A11 8000 CD91 0321 #.#.ß..z.....M..!
17A0 6800 AF06 0477 2310 FC11 1000 1977 115C #h./..w#.ö....w.ö
17B0 000E 13CD 9103 0E16 CD91 03FE FF20 0B0E #...M....M..ß. ..
17C0 0911 2C07 CD91 03C3 DF01 2AEE 0611 0001 #..,.M..C_.*n....
17D0 A7ED 5229 7DB7 2801 244C 2100 010C 0D28 #´mR)ü7(.$L!....(
17E0 1711 8000 0680 7E12 2313 10FA 115C 0041 #......ß.#..z.ö.A
17F0 0E15 CD91 0348 18E6 0E10 115C 00CD 9103 #..M..H.f...ö.M..
1800 212A 0122 2201 2171 0222 3301 212C 0122 #!*.""."!q."3.!,."
```

```
     0 1  2 3  4 5  6 7  8 9  A B  C D  E F

1810 3501 21ED C722 3901 C3DF 0121 960B 1100 #5.!mG"9.C_.!....
1820 C001 850C EDB0 C900 0000 0000 0000 0000 #§...m0I.........
```

Syntaxdiagramm der wichtigsten Befehle und Strukturen

```
---------
I BLOCK I
---------


          -----------------
      ---I   DEKLARATION   I---
     I    ------------------    I
-----I                          I----------------------------Y---->
     I    ------------------    I  ↑   ----------------      |
      ---I ANWEISUNGSTEIL  I---    '---I ANWEISUNGSTEIL I---'
          -----------------            ----------------


 -----------------
I ANWEISUNGSTEIL  I
 -----------------

 ---------     ------------------     -----------------     -----
( PROGRAM )---I EINFACHER ZEIGER I---I ANWEISUNGSLISTE I---( END )
 ---------     ------------------     -----------------     -----


 -----------------
I   DEKLARATION   I
 -----------------

 --------      -------------------     ------
( DECLARE )---I DEKLARATIONSLISTE I---( DEND )
 --------      -------------------     ------


 -------------------
I DEKLARATIONSLISTE I
 -------------------

     -------------------------------------------------------
    I                                                       I
    V     -----        ------------------                   I
-------> ( VAR )-------I EINFACHER ZEIGER I-----------------+----->
    I     -----        ------------------
    I     -------      ------------------     ------        I
    I->  ( CONST )-----I EINFACHER ZEIGER I----I ZAHL I---->I
    I     -------  ↑   ------------------     ------        |
    I     -------  I                                        I
    I->  ( ARRAY )-                                         I
    I     -------                                           I
    I     --------     ------------------     ------        I
     ->  ( STRING )----I EINFACHER ZEIGER I----I TEXT I-----
          --------     ------------------     ------
```

```
 -----------------
I ANWEISUNGSLISTE I
 -----------------

      -----------
-----I ANWEISUNG I----------------------------------Y----->
      -----------      ↑                             |
                       I          -----------        I
                        --------I ANWEISUNG I--------
                                -----------


 ---------
I OPERAND I
 ---------

           ------
----------I ZAHL I---------------Y--------->
    I      ------
    I      ----------            I
    I-->  I AUSDRUCK I-----------I
    I      ----------            I
    I      ------------------    I
     -->  I EINFACHER ZEIGER I---
           ------------------


 ------------------
I EINFACHER ZEIGER I
 ------------------

      ---------
-----I ZEICHEN I----------------------------------Y----->
      ---------      ↑                             |
                     I          ---------          I
                      --------I ZEICHEN I--------
                                ---------


 ---------
I ZEICHEN I
 ---------

           -----------
----------I BUCHSTABE I----------Y--------->
    I      -----------
    I      --------              I
    I-->  I ZIFFER I-------------I
    I      --------              I
    I      ---------------       I
     -->  I SONDERZEICHEN I------
           ---------------
```

```
 ------
I TEXT I
 ------

      ---      ---------                                  ---
-----( ' )---I ZEICHEN I--------------------Y---( ' )--->
      ---      ---------     |                |    ---
                             I   ---------    I
                             ---I ZEICHEN I---
                                 ---------

 ------
I ZAHL I
 ------
      --------
-----I ZIFFER I---------------------------------Y----->
      --------        |                         |
                      I         --------        I
                       --------I ZIFFER I--------
                                --------

 --------
I ZEIGER I
 --------
                ----------------------------------------------------
    ---------  V                                                   I
---I OPERAND I-----------------------------------------------------+->
    ---------   I
                I----------i -----------------i ------------------>I
                I  ------- V I -------------- V I ---------------  I
                 -IOPERANDI---IEINST PRÄDIKATI---IZWEIST PRÄDIKATI-
                   -------     --------------     ---------------

 ----------
I AUSDRUCK I
 ----------
      --------
-----I ZEIGER I-------------------------------------------Y---->
      --------  I                                         ↑
                I    --------     --------------------    I
                 ---I ZEIGER I---I VERGLEICHSOPERATOR I-->I
                I    --------     --------------------    I
                I              ----                       I
                 -------------( 0= )--------------------->
                               ----

 -----------
I BUCHSTABE I          A / B / C ... X / Y / Z
 -----------

 --------
I ZIFFER I             1 / 2 ... 9 / 0
 --------
```

```
 ---------------
I SONDERZEICHEN I     + / - / / / §  ...  ? / # / $
 ---------------
 -----------
I ANWEISUNG I
 -----------
        ----------          --------     ----
---Y---I AUSDRUCK I---Y----I ZEIGER I---( := )----------------------->
   I    ----------    I    ------------
   I    ----           ------------------------------------------->I
   I---( CR )-------------------------------------------------------->I
   I    ----                                                         I
   I                    -------       ------                         I
   I-------------------( WRITE )-----I TEXT I--------------------->I
   I                    -------       ------                         I
   I    --------        -------                                      I
   I---I ZEIGER I--Y---( PRINT )---------------------------------->I
   I    --------   I    -------                                      I
   I               I    ----------                                   I
   I               I---( PRINT(S) )------------------------------->I
   I               I    ----------                                   I
   I               I    ------                                       I
   I               I---( READ )----------------------------------->I
   I               I    ------                                       I
   I               I    ---------                                    I
   I                ---( READ(S) )------------------------------->I
   I                    ---------                                    I
   I    ----------      ----     -----------------      ------       I
   I---I AUSDRUCK I---( IF )-Y-I ANWEISUNGSLISTE I--Y-( ELSE )---   I
   I    ----------      ----  I  ----------------- I I  ------    I I
   I                          --------------------  I            I I
   I                                                I            I I
   I                          ----------------------)------------  I
   I                         I   -----------------  I  ------      I
   I                          ->I ANWEISUNGSLISTE I---( THEN )---->I
   I                             -----------------     ------      I
   I                                                                I
   I                 ------------                                   I
   I    --------    I            I    ----------     -------        I
   I---( REPEAT )-Y-I ANWEISUNGS I---I AUSDRUCK I---( UNTIL )---    I
   I    --------  I I   LISTE    I ↑ ----------     -------     |   |
   I              I  ------------  I                           I   I
   I               ----------------                            I   I
   I                         ----------------------------------    I
   I                        I          ------------                 I
   I                        I         I            I    ------      I
   I                         -------Y-I ANWEISUNGS I---( LOOP )--->I
   I                                I I   LISTE    I ↑ ------      |
   I                                I  ------------  I             I
   I                                 ----------------              I
   I                                       ------------            I
   I    ---------    ---------    -----   I            I  ------   I
   I---I OPERAND I--I OPERAND I--( FOR )--I ANWEISUNGS I-( LOOP )->
        ---------    ---------    -----   I   LISTE    I  ------
                                          ------------
```

Liste der Schlüsselworte und Standardnamen

Schlüsselworte

in alphabetischer Reihenfolge. Die angegebenen Seitenzahlen bezeichnen denjenigen Ort, an dem die Bedeutung des Schlüsselwortes erklärt wird.

AND	35	LOOP	48/51
ARRAY	29	MOD	34
CALL	20	NOT	33
CEND	20	OR	35
CODE	20	PROGRAM	29
CONST	29	REPEAT	48
DIV	34	STRING	29
ELSE	43	THEN	43
END	29	UNTIL	48
FOR	51	VAR	29
IF	43	VARF	56

Standardnamen

in alphabetischer Reihenfolge. Die angegebenen Seitenzahlen bezeichnen denjenigen Ort, an dem die Bedeutung des Standardnamens erklärt wird.

ACTDRV	59	PRINTF	57
CHGDRIVE	60	READ	42
CLOSE	56	READF	57
COMPILE	55	READ(S)	42
CR	41	RND	33
DATEINAME	60	SCROFF	59
DEC	33	SCRON	59
DELETE	59	SWAP	35
DUP	33	TEND	59
INC	32	TREADF	57
NOFILES	59	TWRITEF	58
OPEN	56	WRITE	41
PRINT	41	WRITEF	58
PRINT(S)	41		

Literaturverzeichnis

(1) Barden, William Jr.: The Z-80 Microcomputer Handbook, Indianapolis, 4. Auflage 1980.

(2) Bengtson, Peter: Forth - die Sprache der Achtziger Jahre ?, mc, 1981, Heft 4, Seite 57.

(3) Denert, E./Frank, R.: Datenstrukturen, Zürich, 1977.

(4) Floegel, E.: FORTH Handbuch, Holzkirchen, 1980.

(5) Gardner, Martin: Mathematical Games, Scientific American, 1975 Heft 3 und 4.

(6) Janich, Ing.(grad.) Dietmar: ZDOS Betriebssystem für Z80 - Computer, Benutzerhandbuch Version 5.a, o.O., 1982.

(7) Oberschelp, W./Wille, D.: Mathematischer Einführungskurs für Informatiker, Stuttgart, 1976.

(8) Ottmann, T./Widmayer, P.: Programmieren mit PASCAL, Stuttgart, 1980.

(9) Wirth, N.: Systematisches Programmieren, Stuttgart, 3. Auflage 1978.

(10) Wirth, N.: Algorithmen und Datenstrukturen, Stuttgart, 2. Auflage 1979.

(11) Wirth, N.: Compilerbau, Stuttgart, 2. Auflage 1981.

RPNL auf einen Blick

CODE	:	leitet Prgtyp 2 ein
CALL	:	leitet Prgtyp 1 ein
?	:	legt Variablenwert auf den Stack
?(B)	:	wie '?' mit 1 Byte
:=	:	Zuweisung (2 Byte)
:=(B)	:	Zuweisung (1 Byte)
INC	:	TOP um 1 erhöhen
DUP	:	TOP := TOP-1
DEC	:	TOP um 1 erniedrigen
+	:	TOP := TOP + TOP-1
-	:	TOP := TOP - TOP-1
DIV	:	TOP := TOP/TOP-1
*	:	TOP := TOP*TOP-1
MOD	:	Rest von (TOP/TOP-1)
CR	:	Zeilenvorschub
NOT	:	logische Funktion
AND	:	logische Funktion
OR	:	logische Funktion
PROGRAM	:	leitet Ptgtyp 3 ein
OUTPUT	:	Zeichenausgabe
=	:	log. Vergleichsoperator
0=	:	log. Vergleichsoperator
<	:	log. Vergleichsoperator
>	:	log. Vergleichsoperator
>=	:	log. Vergleichsoperator

<=	:	log. Vergleichsoperator
<>	:	log. Vergleichsoperator
I	:	Laufvariable
SWAP	:	vertauscht TOP und TOP-1
=(S)	:	Vergleichsoperator für Strings
>=(S)	:	Vergleichsoperator für Strings
PRINT(S)	:	Ausgabeprozedur für Strings
READ(S)	:	Eingabeprozedur für Strings
READ(CH)	:	Eingabeprozedur für Strings
INCHAR	:	Eingabeprozedur für Zeichen
OUTCHAR	:	Ausgabeprozedur für Zeichen
PRINT	:	Ausgabeprozedur für Zahlen
NEW	:	Neuinitialisierung des Compilers
TRUE	:	Konstante
FALSE	:	Konstante
DECLARE	:	leitet Deklaration ein
COMPILE	:	Übersetzen einer Datei
SCROFF	:	schaltet Bildschirm aus
SCRON	:	schaltet Bildschirm ein
SYSTEM	:	Rücksprung ins Betriebssystem
SYSFIELD	:	nächste freie Adresse
BTRBSYS	:	Einsprungadresse ins Betriebssystem
ZEICHEN	:	dort steht das Zeichen, daß mit INCHAR eingegeben wurde
DRON	:	schaltet den Drucker ein
DROFF	:	schaltet den Drcuker aus
LIST	:	listet alle Variablen auf
MEMORY	:	zeigt den Restspeicher an

RND	:	erzeugt Zufallszahlen
SAVE	:	speichert compilierte Programme ab
FILES	:	listet das Directory auf
CHGDRIVE	:	Laufwerkwechsel
DATEINAME	:	Ändern eines Dateinamens
READF	:	Einlesen einer Datei
OPEN	:	Öffnen einer Datei
CLOSE	:	Schließen einer Datei
DELETE	:	Löschen einer Datei
WRITEF	:	Sichern einer Datei
PRINTF	:	Ausgeben einer Datei
TWRITEF	:	Schreiben einer Datei
TREADF	:	Lesen einer Datei
TEND	:	Ende des TWRITEF-Modus
ACTDRV	:	aktuelles Laufwerk
NOFILES	:	Ausschalten der Filefunktionen
CEND	:	Abschluß der Typen 1 und 2
IF	:	Beginn der Bedingungsanweisung
THEN	:	Abschluß der Bedingungsanweisung
ELSE	:	Alternative der Bedingungsanweisung
REPEAT	:	Beginn der Wiederholungsschleife
UNTIL	:	Bedingung der Wiederholungsschleife
LOOP	:	Abschluß der Wiederholungsschleife
FOR	:	Beginn der Wiederholungsschleife
END	:	Abschluß des Types 3
WRITE	:	Textausgabeprozedur
READ	:	Eingabeprozedur für Zahlen

VAR	:	Variablendeklaration
VARF	:	Filedeklaration
CONST	:	Konstantendeklaration
ARRAY	:	Felddeklaration
STRING	:	Zeichenkettendeklaration
DEND	:	Abschluß einer Deklaration

ISBN 3—88707—022—4

W.-D. Luther-Verlag

FÜR WISSENSCHAFT UND TECHNIK